Control and Cybernetics
vol. **54** (2025) No. 3
pages: 309-340
DOI: 10.2478/candc-2025-0011

Implementing a combined defuzzification: evaluation of COBYLA and SLSQP*

by

Jörg Verstraete[1,2], Weronika Radziszewska[1] and Jolanta Jarnicka[1]

[1]Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland
[2]Szewalski Institute of Fluid-Flow Machinery, Polish Academy of Sciences, Fiszera 14, 80-231 Gdańsk, Poland.
jorg.verstraete@ibspan.waw.pl
weronika.radziszewska@ibspan.waw.pl.
jolanta.jarnicka@ibspan.waw.pl

Abstract: The combined defuzzification of multiple fuzzy sets under a shared constraint is an extension of traditional defuzzification. The problem concerns defuzzifying multiple fuzzy sets at once, adhering to a constraint that involves their defuzzified values. This specific problem emerged in an application of fuzzy rulebase systems with the goal of regridding spatial data; the constraint stems from the need to keep the total modelled value over a region consistent. Through the introduction of goal functions, the combined defuzzification problem was translated to an optimization problem; the goal functions allow to use objective criteria to evaluate and rank different solutions. Different goal functions for the combined defuzzifier have been presented in the relevant literature, in this contribution, the aim is to evaluate solvers in order to develop a usable implementation of the novel defuzzifier and to verify their stability and performance for the presented problem.

Keywords: constrained defuzzification, combined defuzzification, spatial reasoning

1. Introduction

When dealing with calculations that yield fuzzy sets, usually at one point there will be the need to interpret and defuzzify the fuzzy results. Typically, a defuzzifier takes a single fuzzy set and extracts a crisp value that is considered a

*Submitted: October 2025; Accepted: December 2025

representative value for the set. What constitutes a *representative* is defined by means of the defuzzifier: examples are mean of max (the representative value is the mean of the values with the highest membership grades) and center of gravity (the center of gravity of the fuzzy set is considered most representative).

In a previous work, Verstraete et al. (2024), the present authors introduced the concept of *simultaneous constrained defuzzification*, in which multiple fuzzy sets are defuzzified in such a way that the defuzzified values satisfy a single constraint. In that work, the constraint required the sum of the defuzzified values be equal to a given value. As, in general, many solutions can satisfy the given constraint, a goal function was introduced to provide a mechanism to identify *better* solutions based on additional criteria - four different goal functions were presented to mathematically define what constitutes a *better solution*. Rather than simply scaling the defuzzified values, the approach deffuzifies the different fuzzy sets simultaneously, taking into account these fuzzy sets and the constraints. To verify how it behaves and qualifies as a defuzzifier, typical properties, see Runkler and Glesner (1993), and Van Leekwijck and Kerre (1999), were considered and tested in Verstraete et al. (2024). Here, attention will be concentrated on the implementation aspects of a simultaneous constrained defuzzifier.

The problem of constrained simultaneous defuzzification occurs when using a fuzzy inference system to perform an intelligent spatial disaggregation (see Verstraete, 2017). The solutions have to be defuzzified in order to be usable in further analysis, while the disaggregation has to be such that the aggregated values does not change.

2. Simultaneous defuzzification

2.1. The concept

In general, a defuzzification extracts a crisp value from a fuzzy set, different operators (e.g. mean of max, center of gravity) reflect different interpretations as to what constitutes the most representative crisp value. For the simultaneous defuzzification (Verstraete et al., 2024), we consider n fuzzy sets $A_i, i = 1, ..., n$ over the domain $\mathbb{R}$. The simultaneous constrained defuzzification D' is then defined as $D' : \wp(\mathbb{R})^n \to \mathbb{R}^n, (A_1, ..., A_n) \mapsto (a_1, ..., a_n)$, with $\wp(\mathbb{R})$ being the notation for the fuzzy powerset of $\mathbb{R}$ (set of fuzzy sets on the domain $\mathbb{R}$). In line with the notation, used for the traditional defuzzifiers, we will denote the obtained value for each A_i as $D'(A_i)$. The constraint in the simultaneous defuzzification D' means that the values $D'(A_i)$ satisfy the following equality

$$\sum_{i=1}^{n} D'(A_i) - c = 0 \tag{1}$$

with c being the known constraining value. Rather than directly determining a defuzzified value, the approach will start from the defuzzified values $D(A_i)$ for each fuzzy set A_i and search for a translation x_i, such that $D'(A_i) = D(A_i) + x_i$. The choice of the defuzzifier D (e.g. CoG, Meom) is a parameter in the approach, while the size of the translations x_i is an additional criterion that can be considered in the goal functions. The constraint is therefore rewritten using translations from this known defuzzified value:

$$\sum_{i=1}^{n}(D(A_i) + x_i) - c = 0 \tag{2}$$

with $D(A)$ denoting the defuzzified value of A, obtained using a known defuzzifier. As in Verstraete et al. (2024), the discussion further on will assume that the contribution of each x_i in the constraint is equal:

$$\begin{aligned}&\forall i, j, \forall \epsilon : f(A_1, ..., A_i, ..., A_n, x_1, ...x_i + \epsilon, ..., x_n)\\ &= f(x_1, ...x_j + \epsilon, ..., x_n).\end{aligned} \tag{3}$$

This is true for the above sum-constraint, on which the discussion will mainly be focused.

2.2. Goal functions

To qualify the solutions, four different goal functions were considered. The goal function G.1 matches the concept that solutions with *higher membership* are *better* values. The goal function G.2 implies that *better* solutions are those that are the closest to the values of the underlying defuzzifier; this results in a defuzzifier that works independently of the shape of the membership function, as only the size of the translations matters. It has the side effect consisting in that all translations x_i will be equal. The goal functions G.3 and G.4 combine aspects of G.1 with G.2 by utilizing an area that depends on both the size of the translation and the membership grade.

Thus, the following goal functions $g : \wp(\mathbb{R})^n \times \mathbb{R}^n \to \mathbb{R} : (A_1, ..., A_n, x_1, .., x_n) \mapsto g(A_1, ..., A_n, x_1, .., x_n)$ were introduced:

G.1 $g(A_1, ..., A_n, x_1, .., x_n) = \min_{i=1..n}(A_i(D(A_i) + x_i))$

G.2 $g(A_1, ..., A_n, x_1, .., x_n) = \max_{i=1..n}(|x_i|)$

G.3 $g(A_1, ..., A_n, x_1, .., x_n) = \sum_{i=1}^{n} |\int_{D(A_i)}^{D(A_i)+x_i} 2 - A_i(x)dx|$

G.4 $g(A_1, ..., A_n, x_1, .., x_n) = \sum_{i=1}^{n} |\int_{D(A_i)}^{D(A_i)+x_i} h(x) - A_i(x)dx|$, with $h(x) =$

$$\begin{cases} |x - D(A_i)|/(D(A_i) - x_{\inf}) + 1 & \text{if } x < D(A_i) \\ |x - D(A_i)|/(x_{\sup} - D(A_i)) + 1 & \text{if } x > D(A_i) \\ 0 & \text{if } x = D(A_i) \end{cases}$$

The goal functions G.2, G.3 and G.4 lead to minimization problems: the best solution to the problem is the solution that minimizes the value of the goal function. In G.1, the aim is to maximize the lowest membership grade, which leads to a maximization problem, which easily can be converted to the minimization problem of complements. Independently of the underlying defuzzifier D, the goal functions G.2, G.3 and G.4 are - in each argument x_i - strictly decreasing for negative values, and strictly increasing for positive values. This does not hold for G.1.

3. Implementation

3.1. Brute force

The simplest implementation, which was used in Verstraete et al. (2024), is a brute-force implementation that exhaustively considers a large set of solutions and evaluates the goal function. To achieve this, it iterates over all possible combinations of a discrete number of points of the fuzzy sets. The calculation time, however, grows exponentially with the number of discrete points per fuzzy set and the number of fuzzy sets, limiting the applicability in practice. In addition, the discretization of the search spaces implies that the returned solution will most likely not be an optimum, but merely a solution quite close to it. Brute force does, however, provide for a suitable solution, by which other approaches can be verified, as its application is easy and its behaviour does not depend on the fine tuning of specific parameters. For small experiments and verification, the speed of finding a solution is of little importance. In addition, even with a limited number of discrete points, this approach can quickly provide appropriate initial values that can be used in the numerical solvers.

3.2. Analysis of the problem

Determining n crisp numbers that share a single constraint using an objective function is an *unbound*, *constrained optimization* problem. *Unbound* as no limits are imposed on the domain of the solutions or on the individual solutions themselves, *constrained* as there is a constraint defined that restricts the space of solutions and *optimization* as there is an objective function that needs to be minimized (or maximized in the case of G.1).

The study of the constrained defuzzifier (Verstraete et al., 2024) revealed non-compliance with some interesting properties, such as the definition of per-

missible zones (Runkler and Glesner, 1993), support selection or additional constraints on the membership grades of the results. However, a solver that would allow for the introduction of additional bounds on the domain of solutions can be used to define permissible zones or to force support selection. This would change the problem to a *bound* optimization problem. In general, the possibility of including additional bounds or constraints also permits to specify, e.g., a maximum translation, a minimum membership grade of the solutions or other criteria.

Many solvers make use of the derivative of the objective function. In Section 2.2, four objective functions were introduced, but these are not necessarily the only ones that can be interesting. A solver that does not depend on the use of the derivative of the objective function should be more universally applicable (and thus offer more freedom in the choice of goal function and in making an implementation independent of this choice), most likely at the expense of a slower convergence. When looking at the goal functions, one notices that the derivative for both G.1 and G.2 is somewhat problematic, hence there is a justification for considering a solver that does not rely on it.

Common additional classifications among solvers are the distinction between linear and non-linear constraints, but also the distinction between equality and inequality constraints. The here considered sum-constraint is an example of a linear equality constraint, however, an equality-constraint is easily rewritten as an epsilon-test with a sufficiently small epsilon, permitting the use of solvers that support only inequality constraints.

3.3. Solvers under consideration

Particle Swarm Optimization

A popular approach to numerical problems is Partice Swarm Optimization (PSO), where candidate solutions (particles) are allowed to move in a search space following mathematically defined formulas for adjusting their velocity and direction. Particle Swarm Optimization has the possiblity to converge to a local optimum, but it has been shown that the algorithm needs proper setting of the weight values to balance the particle's best and swarm's best solutions. We considered two implementations of PSO in Python, *PySwarm* and *PyMOO*, as these implementations support constrained problems, and tested the set of examples presented in Section 4.1. The two implementations use different parameters to adjust the behaviour of the swarm: apart from the common aspect such as swarm size, PyMoo allows for setting the iteration used for the velocity update, the cognitive impact and the social impact; PySwarm has parameters such as particle velocity scaling, scaling factors to search away from the best known position and iterations. Both require the lower and upper bound of the

parameters to be established and both allow for optional inequality constraints; PyMoo is further capable of using an adaptive mechanism to improve the search efficiency.

However, in the experiments, we were unable to consistently achieve good solutions for the series of problems with either of the considered libraries, returning solutions that often do not adhere to the imposed constraint. As the implementation should work for different fuzzy sets, goal functions and underlying defuzzifiers, the need for first finding suitable parameters for any problem significantly lowers the usability; for this reason this approach was not pursued further.

COBYLA

COBYLA, which stands for Constrained Optimization BY Linear Approximations, is a derivative-free optimization tool that allows for the non-linear inequality constraints (Powell, 1994). The algorithm is capable of solving non-smooth non-linear programming problems with a moderate number of variables. The COBYLA algorithm converges slowly, due to the use of linear approximations (see Powell, 2012). Owing to the use of quadratic approximations, the algorithms NEWUOA and BOBYQA (Bound Optimization BY Quadratic Approximation), both of these having been also developed by J.M.D. Powell, converge faster than COBYLA, but only work for respectively unbound and bound constrained problems. The original algorithm was written in Fortran77, ported to Fortran90, C, C# and it is present in the Python scientific calculations library *SciPy*, along with the package PDFO (Powell's Derivative-Free Optimization solvers, Ragonneau, 2024), which contains different solvers developped by J.M.D. Powell.

This is a more classical approach to numerical solvers and the first one considered as it meets all requirements. Apart from meeting the basic requirements, COBYLA also allows to satisfy some of the properties that could be fulfilled by adding constraints or boundaries: constraints imposed on the membership grades of the solutions can, for example, be used to search for solutions that comply with support-selection; constraints on the values of the solutions themselves will allow for the definition of permissible zones (Runkler and Glesner, 1993). The ability to work with both bound and unbound problems makes the COBYLA solver suitable under very different goals. We tested three implementations: JCOBYLA - a straight Java port from the C# version, based on the Fortran90 implementation, SciPy, and PDFO. All three behaved quite similarly; the experiments, described further on, were performed using the PDFO implementation (Ragonneau and Zhang, 2024), mainly because it is well maintained and documented.

Table 1: Defuzzified values for the fuzzy sets in the example

	A	B	C	sum
COG	0.3167	0.681	0.3048	1.3025
MeOM	0.25	0.65	0.5	1.4
domain	$[0.0, 0.7]$	$[0.4, 1.0]$	$[0.0, 0.6]$	$[0.4, 2.3]$

SLSQP

The standard Python library SciPy offers various solvers, but few are derivative-free. Foregoing this requirement limits the choice of the goal functions that can be used, but still applies for G.3* and G.4. As such, we also consider the solver SLSQP, Sequential Least SQuares Programming, which requires the objective function to be twice differentiable and works for both bound and unbound constrained problems.

4. Study of the selected solvers

4.1. Description of the example

In order to discuss the performance and compare the behaviour of the solvers, an example involving three fuzzy sets was constructed: a triangular fuzzy set A, defined by $\{(0.0, 0.0), (0.25, 1.0), (0.70, 0.0)\}$, a trapezoidal fuzzy set B, defined by $\{(0.4, 0.0), (0.6, 1.0), (0.7, 1.0), (1.0, 0.0)\}$, and a non-convex fuzzy set C, defined by $\{(0.0, 0.0), (0.1, 0.8), (0.2, 0.7), (0.5, 1.0), (0.6, 0.0)\}$. The fuzzy sets, as well as their values for different goal functions in relation to CoG and MeOM are shown in Fig. 1.

The diagrams of Fig. 1 clearly show that the goal function G.1 is independent of the defuzzifier, and that the goal functions G.2, G.3 and G.4 are decreasing to the left of the defuzzified value and increasing to the right of it. The defuzzified values of these fuzzy sets are listed in Table 1. In addition, the table contains the domain-limits and the sums of defuzzifiers and domain limits. The sum of the domain limits determines the range, for which a solution within the support-intervals of the fuzzy sets exists; the sum of the defuzzifiers shows the constraining value that requires no translation from the underlying defuzzifier, as well as the range within which solutions exist within the supports.

These fuzzy sets will be used to generate a large collection of examples that allow to verify the success rate, but also the stability: a small change in either

*Strictly speaking, the goal function G.3 is not differentiable in the points, where the translation is 0, but as the function satisfies unidirectionality, no non-zero solution will have these points as interior points in the solution space.

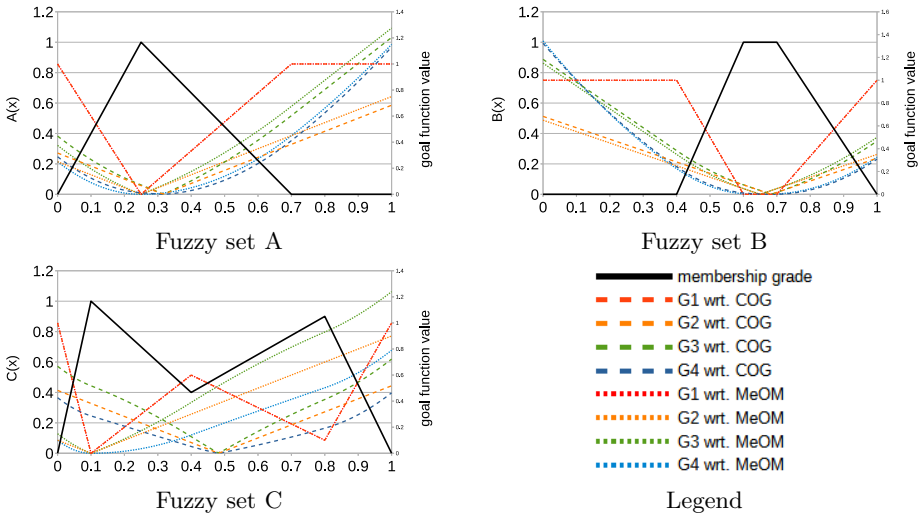

Figure 1: The fuzzy sets used in the example, along with the values of the goal functions relative both to the COG and MeOM defuzzifiers. The membership grades are shown in black using the left y-axis; the goal function values use the right y-axis

the constraining value or a fuzzy set should not produce a big difference in the solution. For this purpose, a sum-constraint with value in the range $[0.3, 2.4]$ will be introduced - some values in this range will lead to solutions outside of the supports.

4.2. Setup of the experiments

The experiments were meant to compare three approaches for solving the simultaneous constrained defuzzification: brute force, SLSQP (SciPY) and COBYLA (PDFO). In the discussion, we do not consider the speed of convergence, as this article is oriented more at the suitability in a specific application.

- Brute force: recursively iterates over the interval $[0, 2.4]$ in 1,000 steps, first for A then for B, and - to speed up the calculation - analytically determines the value of C that fits the constraint (along with its contribution to the goal function). In case of many solutions, the algorithm will return the first solution that met the criteria. As soon as the goal function value in an iteration exceeds that of the current best solution, the search space is pruned, by removing solutions that cannot yield a better goal value, thus speeding up the calculations of the algorithm.

- COBYLA: starts from an initial solution, supplied by brute force (considering 500 values in $[0, 2.4]$), initial radius set to 10^{-2}, final radius set to 10^{-8}, maximum number of number of evaluations set to $100,000$.
- SLSQP: starts from an initial solution, supplied by brute force (considering 500 values in $[0, 2.4]$), solution tolerance set to 10^{-8}, maximum number of iterations being set to $100,000$.

The interval $[0, 2.4]$, used for brute force algorithm and for the initial guesses, is chosen so that all combinations of positive numbers that yield the sum-constraint in any of the experiments are evaluated. The same interval is also used for all experiments. The number of iterations for SLSQP and COBYLA were deliberately set very high, in order to give the solvers ample room in terms of operations to search for the best solution; in none of the tests this limit was reached.

4.3. Experiment 1: stability regarding changes in the arguments

In this experiment, we observe the impact of small changes in a single fuzzy set on the outcome of the constrained defuzzification. Ideally, the difference between the defuzzified values remains small when the difference between the arguments is small. This not only depends on the theoretical analysis, but also on the behaviour of the solver, especially if there are multiple solutions. The constrained defuzzification of the three fuzzy sets, defined before, was performed, using every combination of goal function (G.1, G.1, G.3, G.4) and of the underlying defuzzifier (COG, MeOM). With two solvers (COBYLA, SLSQP) and brute-force, this yields 24 different cases. For each case, 46 tests are run: the fuzzy set A is translated 45 times along the X-axis, in steps of 0.01, while the constraint is kept fixed at 1.6 - this is an arbitrary choice, such that solutions will initially require values to the left of the underlying defuzzifiers, and move to the right as the fuzzy set A is translated to the right [†]. The discussion will start from goal function G.4: this is the case with the least of issues, which makes it easiest to describe what is and what should be visible on the figures.

Goal function G.4

Figure 2 graphically represents the outcome from the set of tests using brute force with underlying defuzzifier COG and goal function G.4. Each individual test is listed along the X-axis, labelled with the size of the translation of the fuzzy set A. For each test, the bars represent the defuzzified values of each fuzzy set (the bars are stacked to visualize the sum referenced to the left axis,

[†] As both CoG and MeOM are translation-invariant, the relative position of the underlying defuzzified value does not change, due to the translation of the fuzzy set; the observed change in the solution of the constrained problem is therefore entirely due to the translation of the fuzzy set.

the result of each test should - and does - sum up to 1.6). The value of the goal function - the contribution of each fuzzy set as well as the aggregated value - is visualized by lines pitted against the right Y-axis. In cases where multiple solutions exist, only the (single) solution returned by the implementation is shown. The presence of multiple solutions for a constraining value and goal-optimum is one source for instability.

The point, where all goal-value functions are 0 is the point where the sum constraint is met without requiring a translation of the underlying defuzzified values. For goal functions that satisfy the unidirectionality, as G.4 does, the goal function and the individual contributions are expected to increase to the left and to the right from this point. Sudden changes in the total goal value should not occur. Changes in the goal function contributions (but not in the total), can be indicative of bigger changes in the returned solutions.

It is clearly visible on the bar charts that translating the fuzzy set A to the right increases its contribution, while decreasing the contributions of B and C. This makes sense as the translation also implies that the value of the underlying defuzzifier increases. The bar charts also indicate that there are no sudden changes between subsequent tests. Comparison of the results obtainrd with SLSQP and COBYLA (Fig. 2) using COG and G4 shows that there are only minor differences. The individual goal contributions fluctuate more in the case of COBYLA (particularly in the interval $[0.04, 0.12]$, but this has virtually no effect on the stability of the returned solution.

The largest observed difference between two subsequent tests occurs for SLSQP (0.0117), compared to 0.0106 for COBYLA and 0.0096 for brute force; the average difference between two subsequent tests is equal, for all cases, 0.0045. There is a slight difference when we change the underlying defuzzifier to MeOM (Fig. 2): SLSQP has now the smallest maximum and average difference. The maximum and average values of the difference between solutions in subsequent test cases are listed further on, in Table 2. It can be argued that the smallest maximum difference indicates a better performance, especially as the average differences are very close. SLSQP is marginally worse than COBYLA for the goal function G4 (with CoG), but outperforms it with MeOM. Brute force shows a slightly lower maximum when using CoG, but may be less interesting from the performance point of view in a general application.

Goal function G.1

The same tests as above were performed using goal function G.1. Due to its requirements, SLSQP was not expected to work well, which was confirmed by the experiments: it exhibits the largest maximum and average difference between subsequent test cases. Rather alarmingly, in each of the tests it claims to have found a solution, but the values of the goal function are clearly not optimized. On the graph of Fig. 4 this is very visible in the behaviour of

both the goal function as well as through the bar-charts. More important in this experiment is the performance of COBYLA, as an alternative to the slower brute-force approach. While minor differences in the goal contributions can be observed, the returned solutions are very similar; even the average and maximum differences between subsequent cases are almost equal (Table 2). For the change of the underlying defuzzifier to MeOM (Fig. 5), the outcome of the experiments turns out to be very similar.

Goal function G.2

Figures 6 and 7 show the results from the tests using goal function G.2. This goal function is easily implemented without a solver as an equal translation of all underlying defuzzified values (not so visible on the bar charts, as they represent the total contribution, rather than the translation from the defuzzifier); the solver analysis in this case is, however, still interesting. SLSQP - despite the fact that the goal function does not meet the requirements - performs similarly to COBYLA; marginally worse than brute force. While the average difference between subsequent solutions is the same for all three solvers (Table 2), brute force shows the highest maximum - performing the worst according to this measure. The results are similar when MeOM is used as the underlying defuzzifier.

Goal function G.3

The results, obtained using the goal function G.3 are surprising (Figures 8 and 9): even though the conditions for SLSQP are met, it behaves the worst of the three approaches, with the highest maximum and average differences; COBYLA, on the other hand, has a much lower maximum difference (Table 2). Upon changing the underlying defuzzifier to MeOM, SLSQP just manages to outperform brute force and COBYLA with lower maximum, but equal average difference.

Summary

Table 2 summarizes the maximum and average differences between subsequent solutions: for each fuzzy set the difference between subsequent defuzzified values was calculated, the maximum was considered for each fuzzy set seperately. The table contains the maximum and average of these three values.

4.4. Experiment 2: stability under changing constraints

In this experiment, we also consider a number of problems, derived from the fuzzy sets defined before, with every combination of goal functions, underlying defuzzifier and solver. Now, the constraining value is allowed to vary in the range defined by the sum of the domain limits ($[0.4, 2.3]$), enlarged to $[0.3, 2.4]$,

Table 2: Comparison of the maximum and average difference between subsequent tests, for all considered cases to test the stability regarding the arguments

	Brute force		SLSQP		COBYLA	
case	max	average	max	average	max	average
C, G1	0.0120	0.0050	0.2450	0.0332	0.0119	0.0050
M, G1	0.0120	0.0050	0.2450	0.0332	0.0119	0.0050
C, G2	0.0072	0.0044	0.0091	0.0044	0.0091	0.0044
M, G2	0.0072	0.0044	0.0076	0.0045	0.0087	0.0044
C, G3	0.0312	0.0071	0.0558	0.0077	0.0288	0.0073
M, G3	0.0120	0.0050	0.0114	0.0050	0.0144	0.0050
C, G4	0.0096	0.0045	0.0117	0.0045	0.0106	0.0045
M, G4	0.0096	0.0046	0.0082	0.0046	0.0098	0.0047
C: Center of Gravity, M: Mean of Maximum						

in order to also provide for the situations where solutions do not exist within the support. Using steps of 0.02, this amounts to 106 cases, which allows to judge how sensitive the results are with respect to small changes of the constraint. No additional constraints were imposed on the solvers.

The figures are similar to those from the first experiment; the x-axis again matches with the different tests, its value now indicates the magnitude of the constraining value. For each test, there are three stacked bars (the three calculated defuzzified values), as well as the contribution of each solution to the goal function and the aggregated goal function, with reference to, respectively, the left Y-axis and the right Y-axis. For the same reason as in the previous experiment, the discussion will start from goal function G.4.

Goal function G.4

Goal function G.4 meets all requirements for COBYLA and SLSQP. In this experiment, we are looking for similar issues as before: sudden changes in the goal function, and even more sudden changes in the defuzzified values that are indicative of the fact that there are potential issues with the stability of the implementation regarding the constraining value (Fig. 10).

First, there is an interesting difference between tests for values below 0.4: SLSQP finds solutions where $D(A) < 0$. The brute force implementation does look for solutions outside of its loop, but only iterates from 0 to 2.4. Brute force was not adjusted for this, while SLSQP and COBYLA have no domain limits imposed. Both SLSQP and COBYLA employ a variant of brute force with less iterations to determine starting values; SLSQP is perfectly capable of finding solutions outside of the range considered by brute force. COBYLA does

return some negative values for $D(A)$, but it does not fully manage to escape the starting values. However, the small difference between the total goal value in the case of COBYLA and SQSQP (1.010763379 - 0.997610667 = 0.013152712) to some extent justify the algorithm in classifying it as good solution. When we look back at the previous experiment, goal function G1 and COG, the brute force case yields a stepwise function for $D(C)$. The size of this step correlates with the size of the steps in the brute force approach, but COBYLA manages to overcome it, as the goal function values are sufficiently different.

In the current experiment, SLSQP outperforms both COBYLA and brute force, having lower maximum difference and lower average difference, but the latter only by a fraction. Changing the underlying defuzzifier to MeOM leads to the same behaviour, but the difference between SLSQP's maximum difference and the other two is greater (Fig. 11). The results are summarized in Table 3.

Goal function G.1

Goal function G1 does not satisfy the requirements for SLSQP, and it shows (see Fig. 12) that the results of the different tests vary wildly - even when solution within the support exist; the goal is not optimized. Brute force and COBYLA manage and their graphs are similar; the maximum and average differences are also very close. Both show quite a big difference around the 1.06 mark, where the contribution of B suddenly decreases, while that of C increases. This is the result of equally evaluated solutions, due to the trapezoidal membership function B. Near 2.3, defuzzified values for A and B are 0 and only C contributes; this is a logical consequence of using G.1: if no solution exists within the support, then any solution outside it is equally acceptable. This explains why very small changes in values x_i can yield very big differences in the obtained solutions. The use of MeOM, see Fig. 13, does not change the outcome of this case. Table 3 summarizes the maximum and average differences.

This is the first case where there clearly is no stability, in none of the solvers.

Goal function G.2

While the goal function G.2 does not satisfy the requirements for SLSQP, it somehow manages, but is outperformed by COBYLA (lower maximum and average) and even brute force (see Figs 14 and 15). As before, brute force is limited by its implementation and does not return the solution with negative values for $D(C)$. Both COBYLA and SLSQP overcome this, with SLSQP showing some fluctuations in the goal function. COBYLA - whose goal function values are the most stable - has the lowest maximum and average. COBYLA performs worse when the underlying defuzzifier is changed to MeOM. The specific values are listed in Table 3.

Goal function G.3

The impact of the starting values is clearly visible in COBYLA, and to

Table 3: Comparison of the maximum and average difference between subsequent tests, for all considered cases, to test the stability behaviour regarding small changes of the constraint function.

	Brute force		SLSQP		COBYLA	
case	max	average	max	average	max	average
C, G1	1.7024	0.0219	2.469e7	3.990e5	1.7024	0.0212
M, G1	1.7024	0.0219	2469e7	3.99e5	1.7024	0.0212
C, G2	0.0216	0.0066	0.0234	0.0068	0.0095	0.0066
M, G2	0.0120	0.0066	0.0085	0.0066	0.0250	0.0070
C, G3	0.0560	0.0075	0.0478	0.0082	0.0584	0.0076
M, G3	0.0368	0.0069	0.0364	0.0071	0.0344	0.0069
C, G4	0.0216	0.0066	0.0200	0.0066	0.0355	0.0067
M, G4	0.0216	0.0066	0.0162	0.0066	0.0476	0.0070
C: Center of Gravity, M: Mean of Maximum						

a lesser extent for SLSQP (Fig. 16). The latter returns negative values for $D(A)$, but the differences in the total goal function are small. Above the 2.3 mark, the solutions returned by SLSQP differ from those returned by COBYLA, with one favouring contributions of A and the other of C. The reason for this difference is visible in the goal function contribution, but the difference between the total goal values is minimal (1.636739717 for SLSQS at 2.4 vs 1.636739699 for COBYLA at the same point, Table 3). This indicates an issue with the goal function not properly differentiating between different solutions; the results returned by COBYLA are closer to those obtained with brute force. Either case, however, does not exhibit big differences between subsequent tests; SLSQP has the smallest maximum difference, while the smallest average difference is similar. Changing the underlying defuzzifier to MeOM, see Fig. 17 yields very similar conclusions.

Summary

Table 3 summarizes the maximum and average differences between subsequent solutions: for each fuzzy set the difference between subsequent defuzzified values was calculated, the maximum was considered for each fuzzy set seperately. The table contains the maximum and average of these three values.

4.5. Success rate

The two experiments performed provide a large number of examples, which can also be used to assess and discuss the success rate of the solvers. The

implemented solvers have the ability to return whether or not a solution was found; for the cases, for which the solver claimed not to have found a solution, we took a closer look at the returned solution: a solver may be oscillating between two equivalent solutions. The solver in such a case may not converge, but the solution returned can still be good from the defuzzification perspective.

Even though not all goal functions meet the requirements for SLSQP, the solver always returned a solution as found, despite clearly failing to optimize some of the tests (especially with G.1). In a few cases, COBYLA reported that it did not manage to find a solution within the allowed number of evaluations (100,000) - most solutions were found with less than 100 evaluations, while some needed a few thousand. Given this large difference, the solver was probably unable to converge further and started oscillating between solutions. Allowing for more evaluations would therefor not make it converge successfully to a solution. Despite the lack of a reported success, the returned solution did not deviate significantly from the other cases, and did not cause a big difference between subsequent tests, further confirming that the solver oscilated around a good solution.

4.6. Speed of convergence

For the different solvers, the number of iterations or evaluations was logged as a measure for the execution time. The brute force implementation requires the biggest number of iterations, despite the fact that it is (somewhat) optimized by pruning the search space on the basis of the goal function value. For all tests in the second experiment, on average, it needed just over 19 000 iterations. For the same tests, SLSQP needed only 132 evaluations on average, whereas COBYLA needed close to 11 000 evaluations. This high number is to some extent explained by the fact that the average is increased by a small number of tests that took the maximum of 100,000 evaluations before returning a failure of the solver. However, due to its inner workings, COBYLA is known to converge slower. In this application, though, it converges fast enough for the method to be usable: where SLSQP needs between 50-150 iterations for most cases, COBYLA requires closer to 100-400 iterations.

5. Conclusion

In this article, different approaches for dealing with the constrained defuzzification problem using solvers were presented and analyzed. The main focus was on determing whether the use of solvers, in particular SLSQP and COBYLA, results in a stable and predictable behaviour of the defuzzifier. This was examined for the different goal functions and underlying defuzzifiers, using a series

of tests. The tests yielded interesting observations.

1. Stability is less of a problem of the considered solvers (COBYLA, SLSQP), but rather of the goal function.
2. Proper setting of the starting values is important, brute force can help but needs to be well defined as the experiments show that some solvers have issues in finding solutions too far from the starting values.
3. Brute force implementation should made so as not to eliminate extreme candidate solutions.

The article concerned general solvers, but some specific combinations, regarding the problem setting, can be solved much more efficiently: for the goal function G.2, iterations and solvers can be avoided completely by implementing an equal translation in the same direction; for G.1 with only two fuzzy sets, it is possible to sum up fuzzy sets (Zadeh's extension principle) and find the constraining value. Future work will be aimed at developing efficient custom algorithms for the more interesting goal functions (G.3 and G.4).

References

Leekwijck, W. V. and Kerre, E. E. (1999) Defuzzification: criteria and classification. *Fuzzy Sets and Systems*, **108**, 2, 159–178.

Powell, M. J. D. (1994) A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In: S. Gomez and J.-P. Hennart, eds., *Advances in Optimization and Numerical Analysis: Mathematics and Its Applications*. Dordrecht, Springer Netherlands, 51-67. http://dx.doi.org/10.1007/978-94-015-8330-5_4

Powell, M. J. D. (2012) On the convergence of trust region algorithms for unconstrained minimization without derivatives. *Computational Optimization and Applications*, **53**, 527–555.

Ragonneau, T. M. and Zhang, Z. (2024) PDFO: a cross-platform package for Powell's derivative-free optimization solvers. https://pdfo.net/ Accessed: August 2023.

Runkler, T. A. and Glesner, M. (1993) A set of axioms for defuzzification strategies towards a theory of rational defuzzification operators. *[Proceedings 1993] Second IEEE International Conference on Fuzzy Systems*, **2**, 1161–1166.

Verstraete, J. (2017) The Spatial Disaggregation Problem: Simulating Reasoning Using a fuzzy Inference System. *IEEE Transactions on Fuzzy Systems*, **25** (3), 627–641.

Verstraete, J., Radziszewska, W., Kaczmarek-Majer, K. and Bykuć, S. (2024) Combined defuzzification under shared constraint. *IEEE Transactions on Fuzzy Systems*, **32**, 5, 3049–3058.

Figure 2: Stability of the considered solvers with respect to changes in the arguments for goal function G4. Center of Gravity

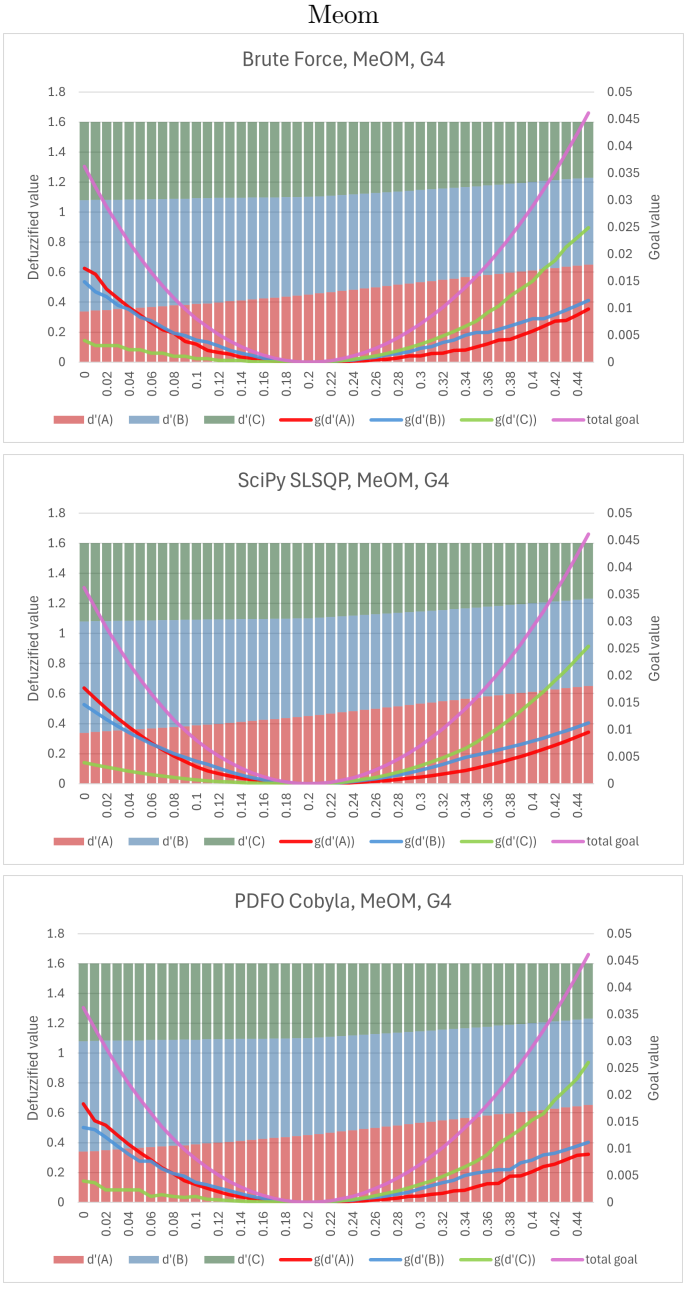

Figure 3: Stability of the considered solvers with respect to changes in the arguments for goal function G4. Meom

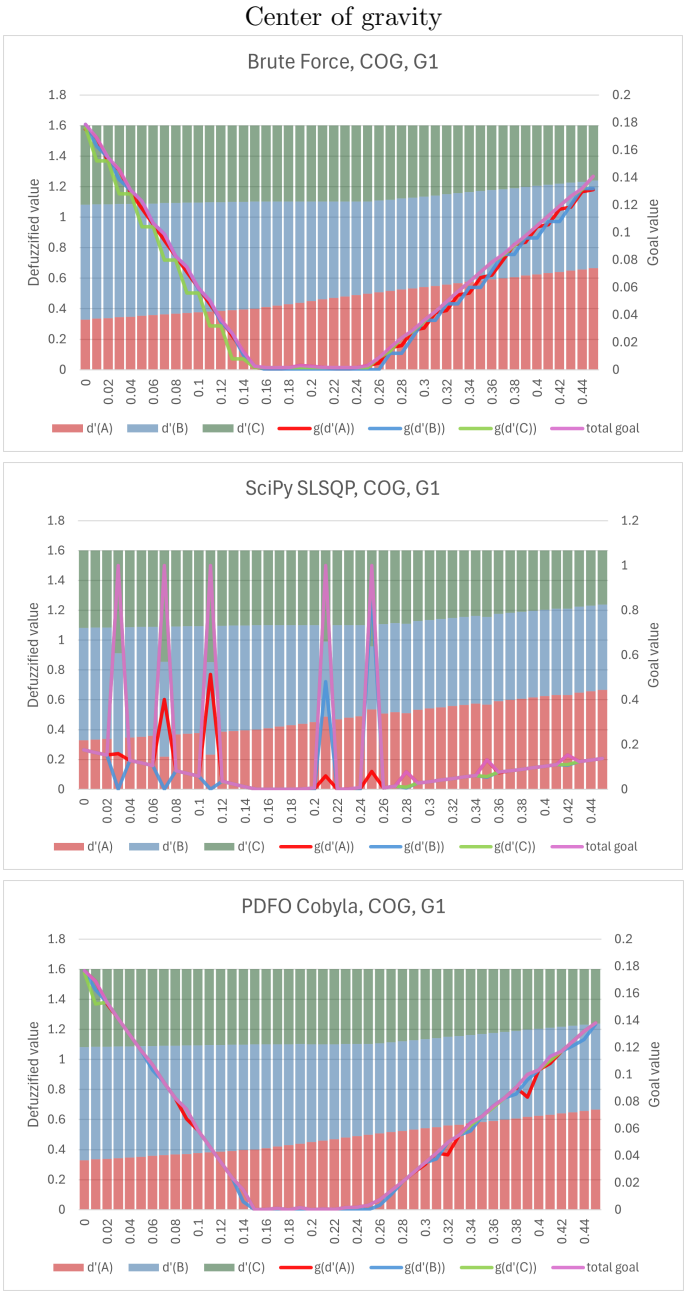

Figure 4: Stability of the considered solvers with respect to changes in the arguments for goal function G1. Center of Gravity Meom

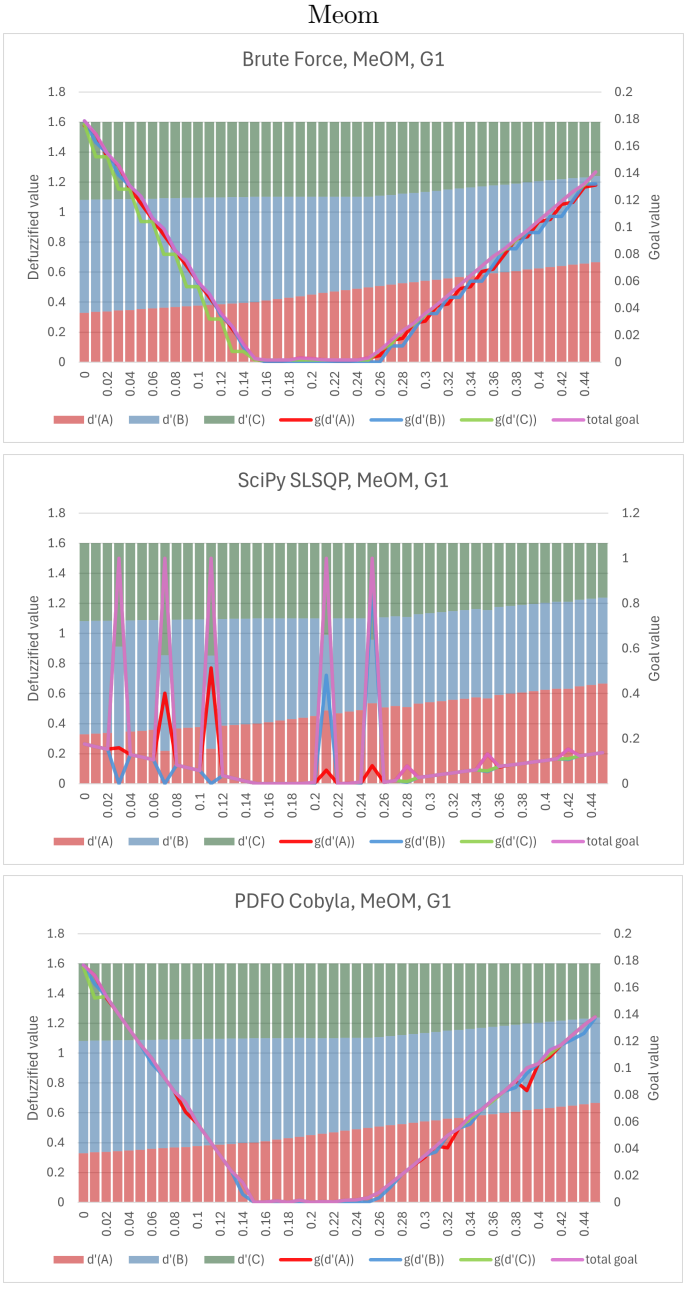

Figure 5: Stability of the considered solvers with respect to changes in the arguments for goal function G1. Meom

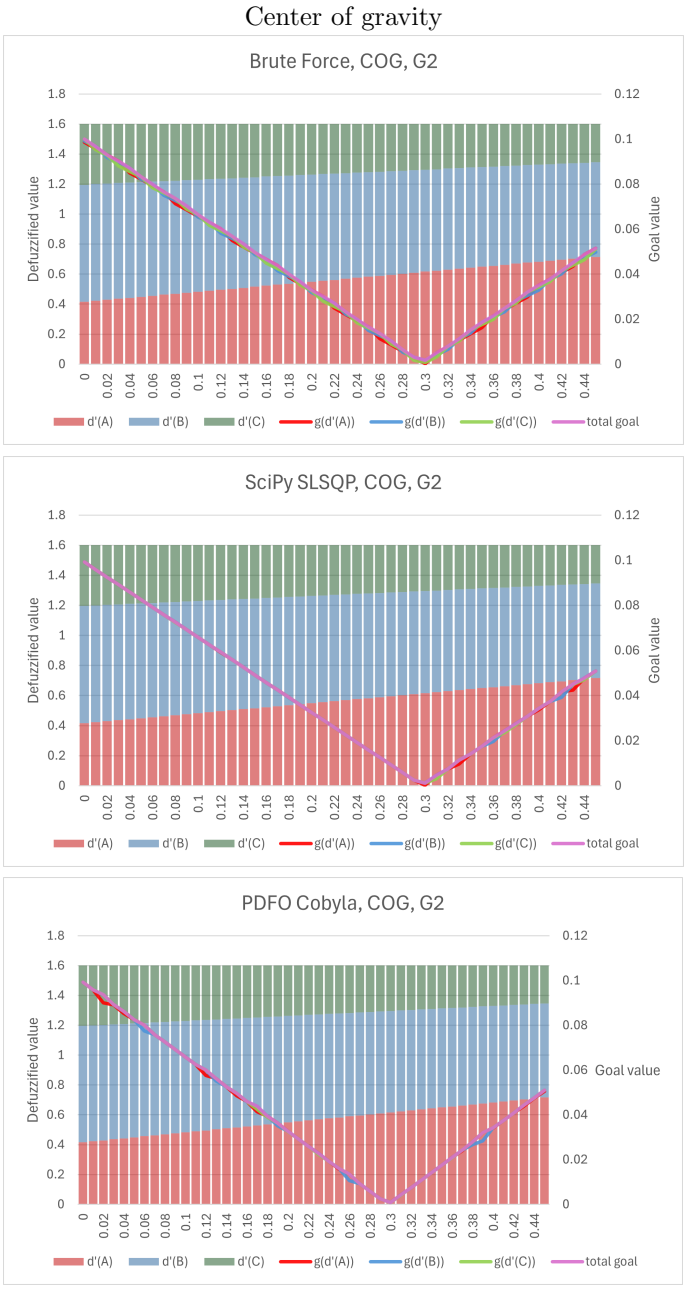

Figure 6: Stability of the considered solvers with respect to changes in the arguments for goal function G2. Center of Gravity

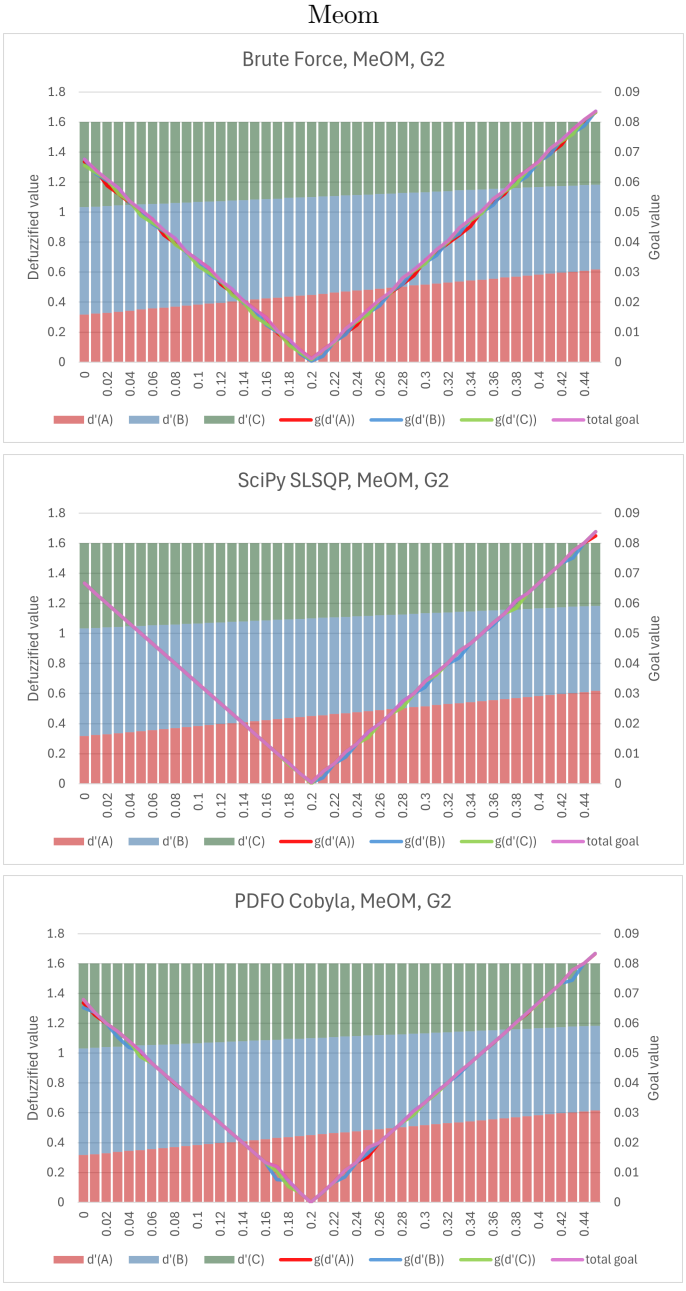

Figure 7: Stability of the considered solvers with respect to changes in the arguments for goal function G2. Meom

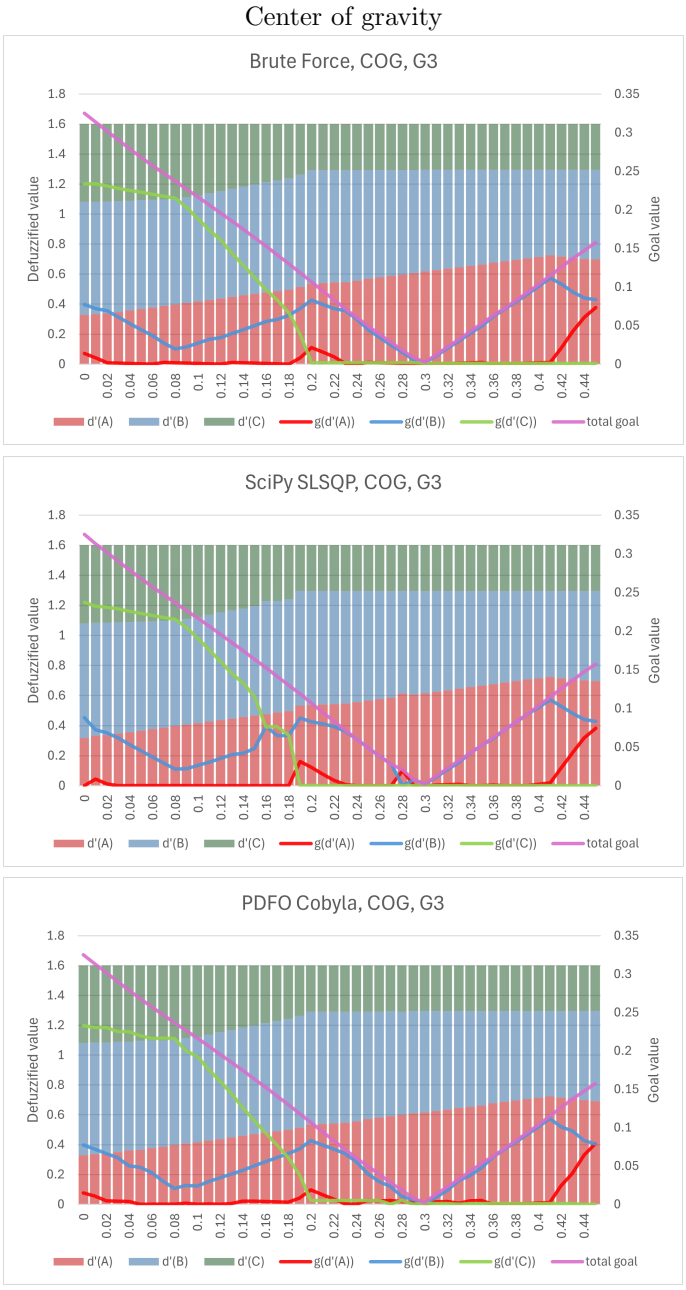

Figure 8: Stability of the considered solvers with respect to changes in the arguments for goal function G3. Center of Gravity

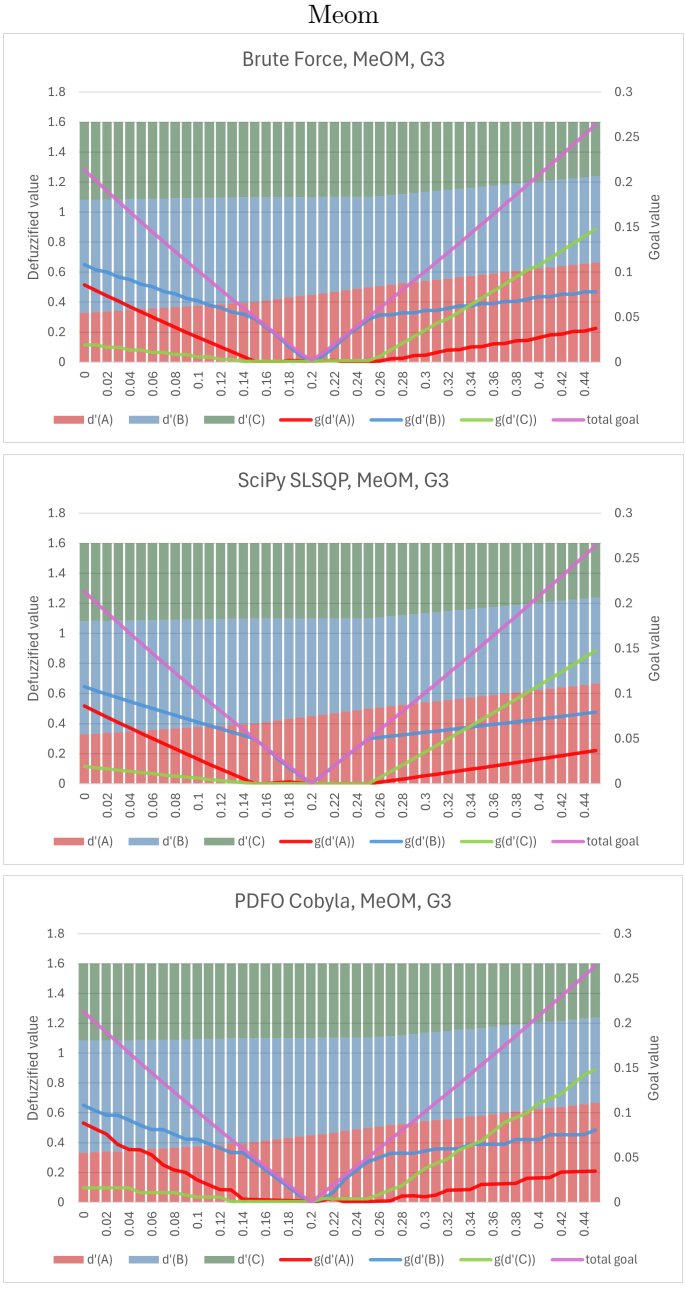

Figure 9: Stability of the considered solvers with respect to changes in the arguments for goal function G3. Meom

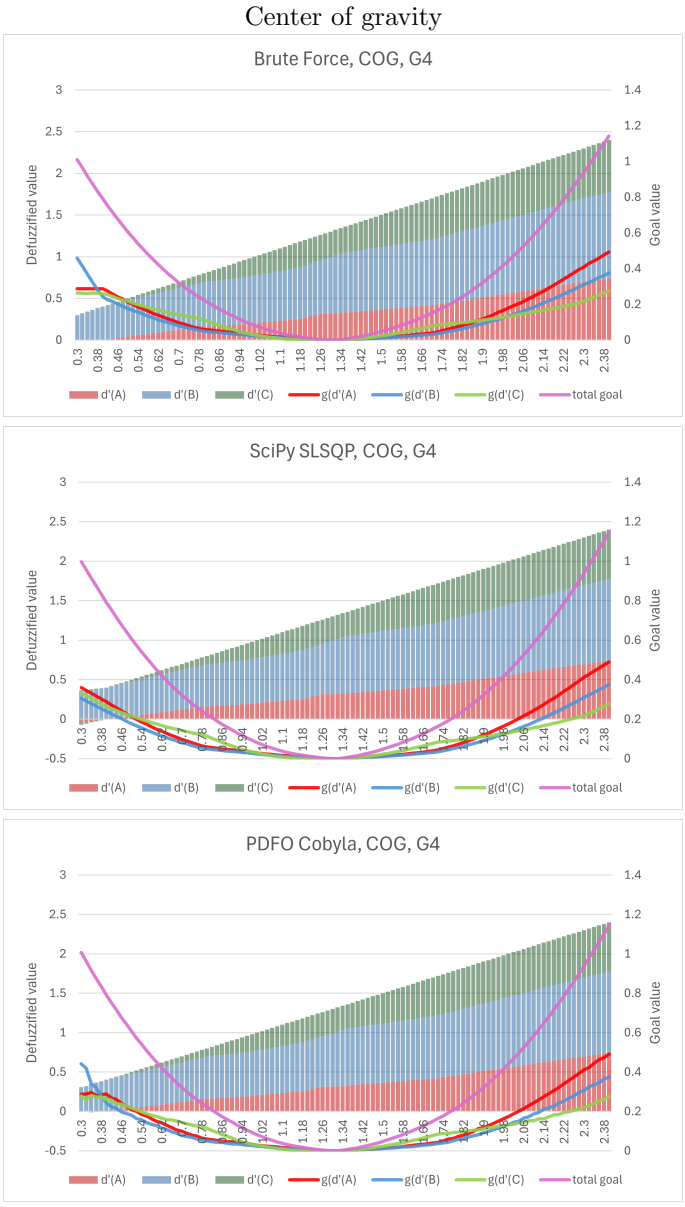

Figure 10: Stability of the considered solvers in respect to changes in the constraint for goal function G4. Center of Gravity

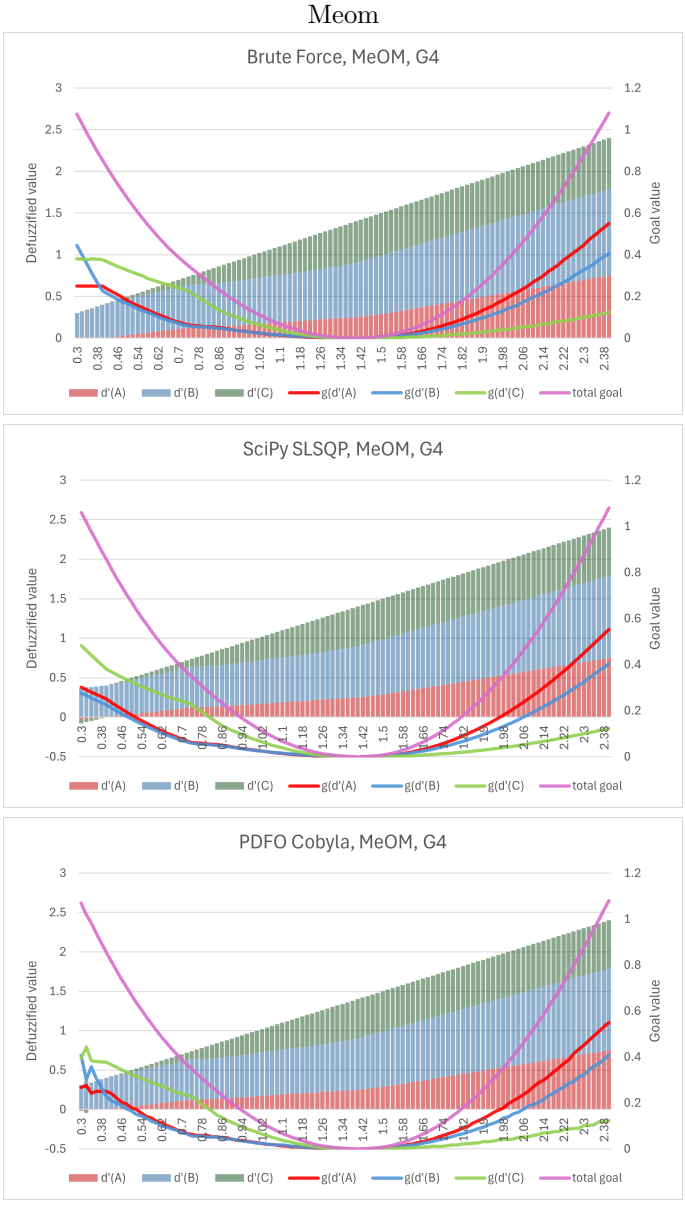

Figure 11: Stability of the considered solvers in respect to changes in the constraint for goal function G4. Meom

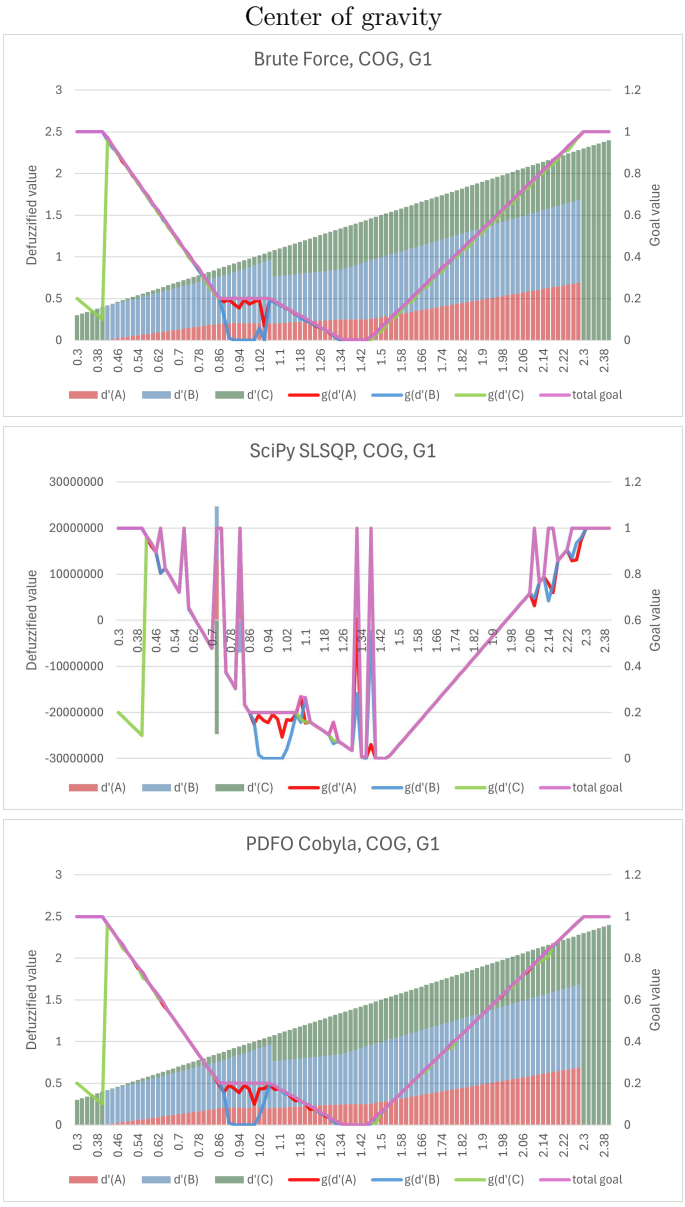

Figure 12: Stability of the considered solvers in respect to changes in the constraint for goal function G1. Center of Gravity

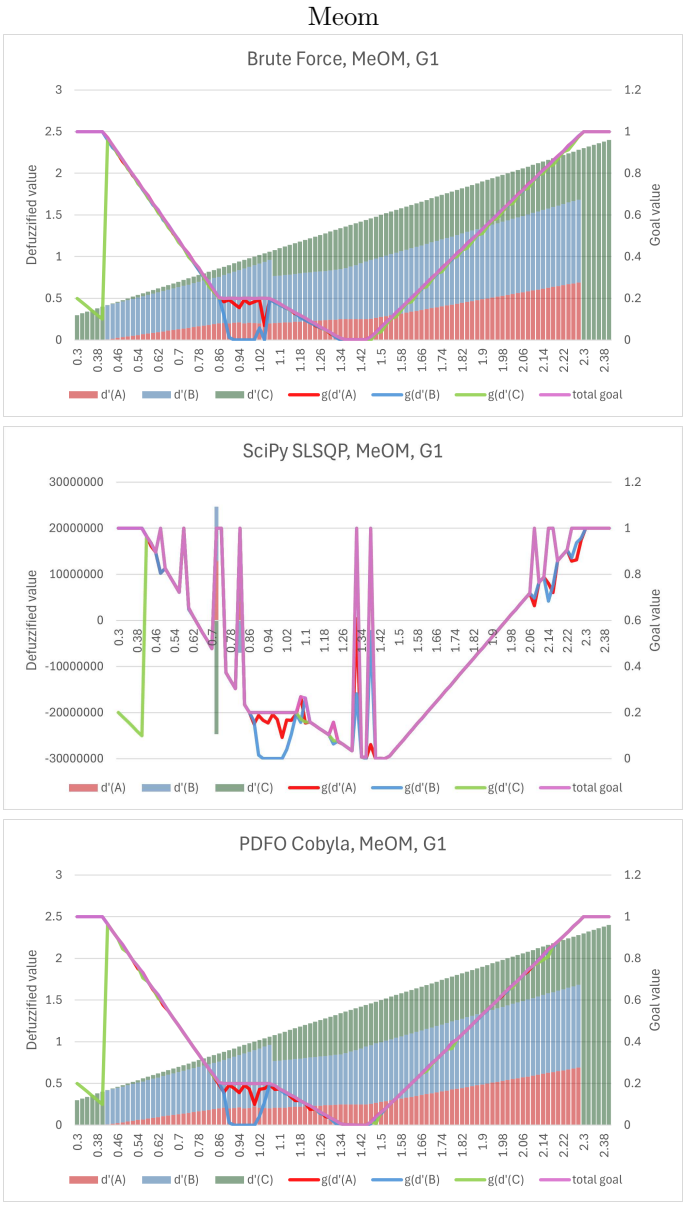

Figure 13: Stability of the considered solvers in respect to changes in the constraint for goal function G1. Meom

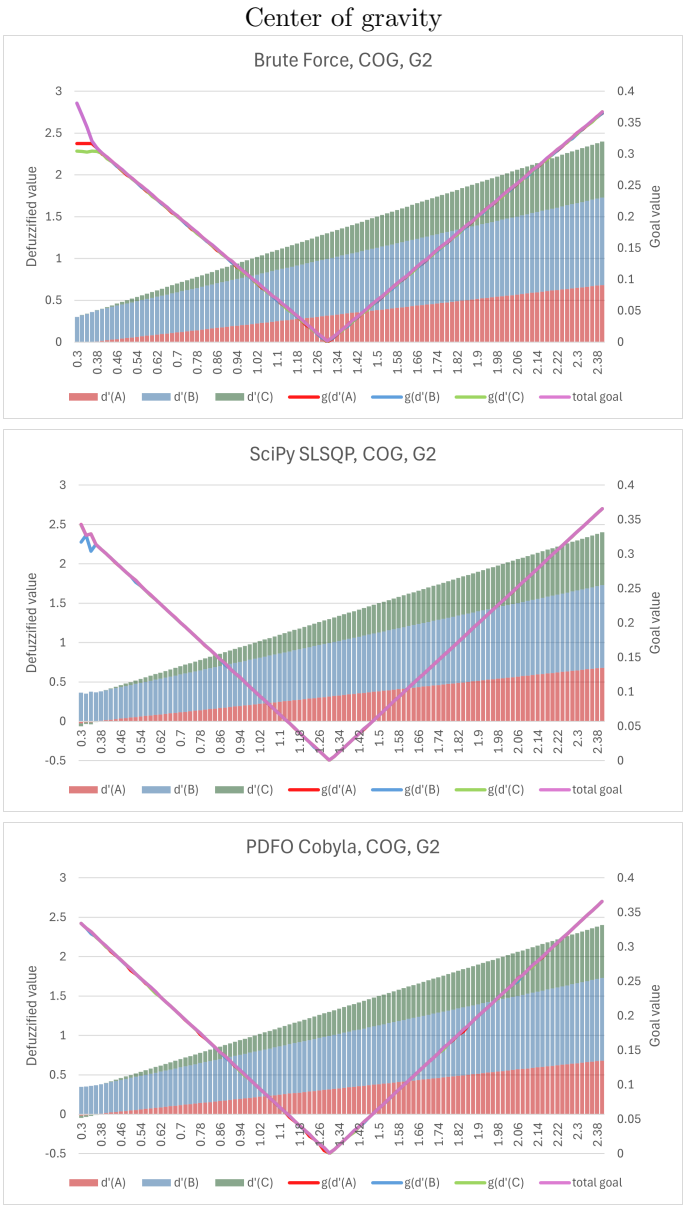

Figure 14: Stability of the considered solvers in respect to changes in the constraint for goal function G2. Center of Gravity

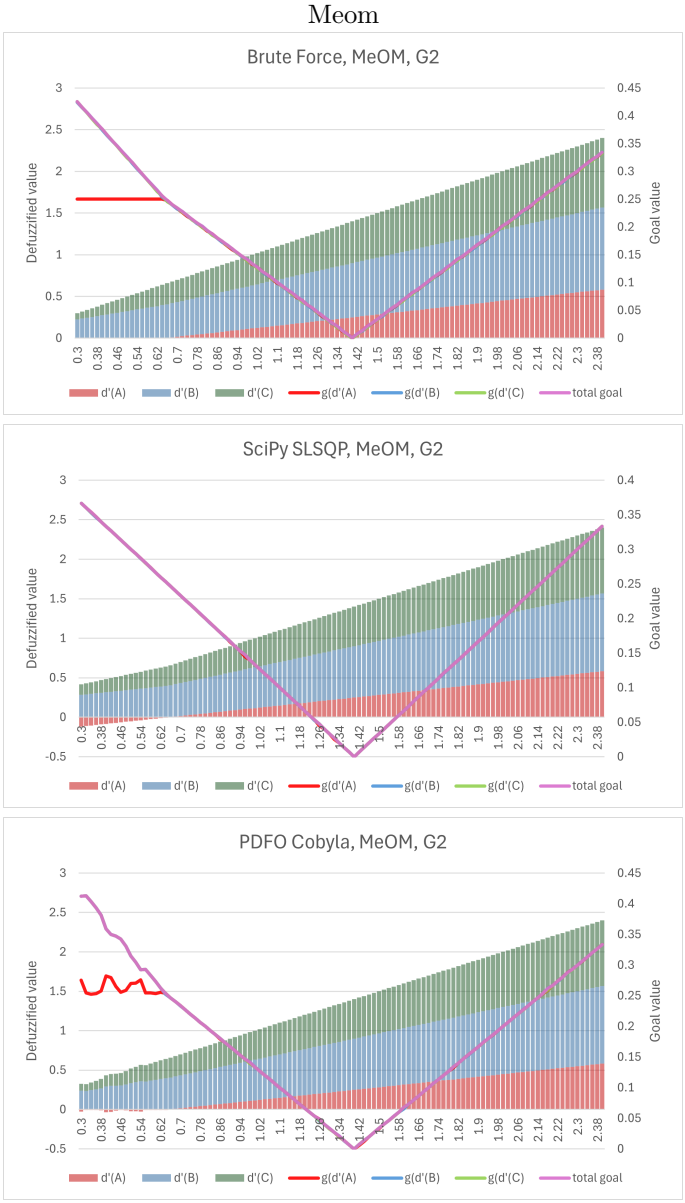

Figure 15: Stability of the considered solvers in respect to changes in the constraint for goal function G2. Meom

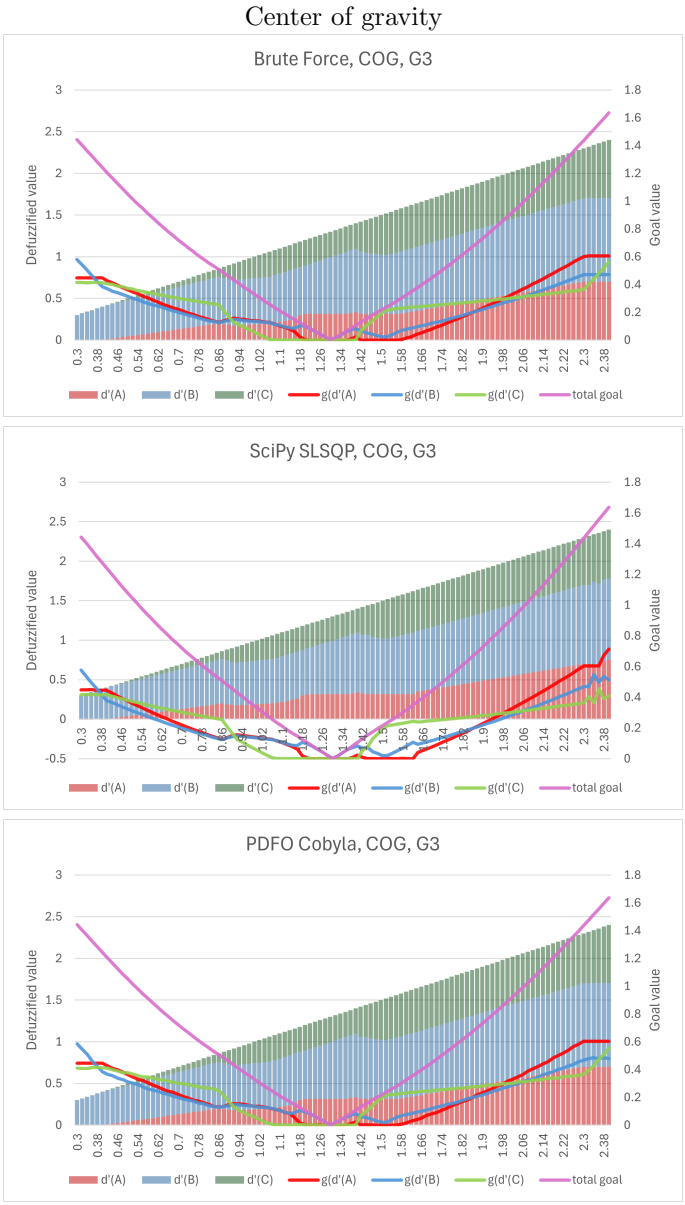

Figure 16: Stability of the considered solvers in respect to changes in the constraint for goal function G3. Center of Gravity

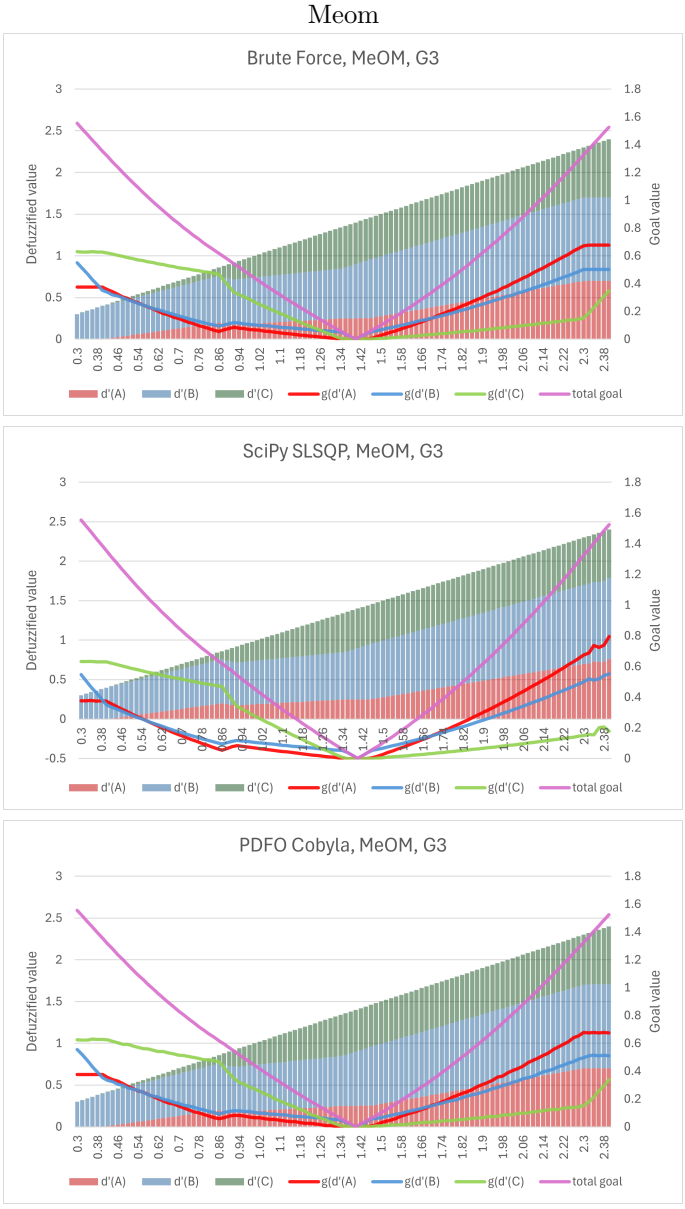

Figure 17: Stability of the considered solvers in respect to changes in the constraint for goal function G3. Meom