Parallelization Strategies for Evolutionary Algorithms for MINLP

Martin Schlueter
Information Initiative Center
Hokkaido University Sapporo
Sapporo 060-0811, Japan
Email: schlueter@midaco-solver.com

Masaharu Munetomo
Information Initiative Center
Hokkaido University Sapporo
Sapporo 060-0811, Japan
Email: munetomo@iic.hokudai.ac.jp

***Abstract*— Two different parallelization strategies for evolutionary algorithms for mixed integer nonlinear programming (MINLP) are discussed and numerically compared in this contribution. The first strategy is to parallelize some internal parts of the evolutionary algorithm. The second strategy is to parallelize the MINLP function calls outside and independently of the evolutionary algorithm. The first strategy is represented here by a genetic algorithm (arGA) for numerical testing. The second strategy is represented by an ant colony optimization algorithm (MIDACO) for numerical testing. It can be shown that the first parallelization strategy represented by arGA is inferior to the serial version of MIDACO, even though if massive parallelization via GPGPU is used. In contrast to this, theoretical and practial tests demonstrate that the parallelization strategy of MIDACO is promising for cpu-time expensive MINLP problems, which often arise in real world applications.**

Keywords: Mixed Integer Nonlinear Programming (MINLP), Ant Colony Optimization (ACO), Genetic Algorithm (GA), MIDACO, Parallelization, GPGPU, Cloud Computing

I. Introduction

Mixed integer nonlinear programming (MINLP) is a research field of growing interest in the evolutionary computing community (see for example Liang et al. [9], Wasanapradit et al. [17], Young et al. [19], Yiqing et al. [18] or Deep et al. [3]). A mathematical formulation of a general MINLP is given in (1), where $f(x, y)$ denotes the objective function to be minimized. In (1), the equality constraints are given by $g_{1,...,me}(x, y)$ and the inequality constraints are given by $g_{me+1,...,m}(x, y)$. The vector x contains the continuous decision variables and the vector y contains the discrete decision variables. Furthermore, some box constraints as x_l, y_l (lower bounds) and x_u, y_u (upper bounds) for the decision variables x and y are considered in (1).

$$\begin{aligned} \text{Minimize} \quad & f(x,y) && (x \in \mathbb{R}^{n_{con}},\ y \in \mathbb{Z}^{n_{int}}) \\ \text{subject to:} \quad & g_i(x,y) = 0, && i = 1,...,m_e \in \mathbb{N} \\ & g_i(x,y) \geq 0, && i = m_e+1,...,m \in \mathbb{N} \\ & x_l \leq x \leq x_u && (x_l,\ x_u \in \mathbb{R}^{n_{con}}) \\ & y_l \leq y \leq y_u && (y_l,\ y_u \in \mathbb{N}^{n_{int}}) \end{aligned} \tag{1}$$

Parallel computing is also a field of rapidly growing interest in the optimization community. With the nowadays widely availability of many core processors, graphic cards with general purpose computing capabilities, computer clusters or cloud computing resources, the question of efficient parallelization strategies for optimization algorithms arises.

The motivation behind this work is to combine the topics of MINLP and parallelization in the context of evolutionary computing. In this contribution two different parallelization strategies for evolutionary algorithms on MINLP should be discussed and evaluated. This contribution does not aim on giving a comprehensive overview on such wide topic, instead it should give some first indication of what can be a promising parallelization strategy for an evolutionary algorithms on MINLP and what not. It appears that the parallelization approaches for evolutionary algorithms (EA) can basically be classified into three strategies: (A) Parallelization of the internal parts of the EA, (B) Executing several instances of EA's in parallel and (C) Executing several problem evaluation in parallel.

As the topic of evolutionary algorithms for MINLP is still a young field, the currently available contributions of parallelized evolutionary algorithms for MINLP remains very small. To the best knowledge of the authors, there are currently two reports about genetic algorithms (Powell and Hollingsworth [10] and Munawar [11]) and one about ant colony optimization (Schlueter [16]) which evaluate the performance of EA's for MINLP in parallel mode.

In this contribution the (massively) parallelized GA named "arGA" from Munawar [11] is compared with results obtained by MIDACO by Schlueter [16]. The code arGA implements the parallelization strategy *A*, while MIDACO implements the parallelization strategy *C*. From the results presented here it can be deduced that strategy *A* is not a promising option, neither for computational cheap MINLP benchmark functions nor for cpu-time intensive MINLP real world applications. In contrast to this, theoretical numerical results show that MIDACO's strategy *C* can be promising if the MINLP is cpu-time expensive. In order to demonstrate this claim, a parallelization case study of MIDACO for a real world MINLP from the area of chemical engineering is presented in addition.

This paper is structured as follows: In Section II, a comparison between the serial and parallelized version of arGA and the serial version of MIDACO is given for a set of eight MINLP

benchmark problems. In Section III, MIDACO's parallelization strategy is evaluated theoretically on the same set of eight MINLP benchmarks. In Section IV, a parallelization case study of MIDACO on a cpu-time intensive real world MINLP application is presented in order to demonstrate its practical usefulness of this approach. Finally some conclusions are drawn.

II. Comparison of parallelized GA with serial MIDACO

In this section a comparison between a Genetic Algorithm (GA) and an Ant Colony Optimization (ACO) algorithm for MINLP is given. The considered GA is namely the implementation arGA (adaptive resolution Genetic Algorithm) by Munawar [11] and the considered ACO is namely MIDACO (Mixed Integer Distributed Ant Colony Optimization) by Schlueter [16]. Both algorithms claim to be advanced ones and implement the oracle penalty method (see Schlueter and Gerdts [14]) for handling constraints. Both implementations are capable of parallelization, but differ in their concrete parallelization strategy. The arGA implementation allows the parallelization of specific GA internal operators. This means the parallelization strategy of arGA aims on reducing the algorithmic overhead of the GA. In contrast to this, MIDACO allows the parallel execution of the problem functions (objective and constraint functions), but does not perform any parallelization of the internal parts of the ACO. This means, the parallelization strategy of MIDACO aims on reducing the calculation times required by the MINLP application, which is often (very) time consuming for real world problems. In summary it can be said, that arGA and MIDACO are both evolutionary algorithms which feature fundamentally different parallelization approaches.

In Munawar [11] results by arGA for a set of eight (small) MINLP benchmark problems is presented. A detailed description of the eight benchmark problems with all relevant data can be found in the Appendix. Note that this selection of benchmark problems is based on the previous work by Munawar [11] and in order to compare the algorithms we are bound to this set here. A comprehensive evaluation of MIDACO on 100 MINLP benchmark problems is alternatively available in Schlueter et al. [15]. In Munawar [11] results are reported for the serial execution of arGA as well as for different parallelization strategies. The parallelization strategy with the best results was reported for GPGPU (using single precision). Details on the parallelization approach by arGA can be found in Munawar [11]. Therefore only two cases for arGA are considered here: the serial case and the GPGPU-parallelization case. Firstly, the results of serial arGA are compared with serial MIDACO in Table I. Secondly, the results of GPGPU parallelized arGA are compared with serial MIDACO in Table II. In Table I and Table II the number of global optimal solutions out of 30 test runs for each benchmark problem is reported together with the average number of function evaluation and the average cpu-time in seconds. Additionally the speedup factor for MIDACO in contrast to arGA is reported regarding the average number of function evaluation and the average time. All test runs (for both arGA and MIDACO) were performed on a computer with Intel Core i7 CPU Q920@2.67GHz. Note, that the identical physical computer was used for both algorithms. Therefore the cpu-run-times are directly comparable.

TABLE I: Comparison of **serial** arGA with serial MIDACO

	arGA			MIDACO			Speed-Up	
Nr.	Optimal	Eval	Time	Optimal	Eval	Time	Eval	Time
1	30	1976	0.21	30	1803	0.0011	1.09	190
2	30	11301	1.38	30	9593	0.0068	1.17	202
3	30	24253	2.87	30	21047	0.0171	1.15	167
4	21	14381	1.48	30	2521	0.0018	5.70	822
5	25	41626	5.92	30	26367	0.0248	1.57	238
6	17	26092	3.21	30	483	0.0004	54.02	8025
7	16	92646	10.86	30	101797	0.1934	0.91	56
8	16	73581	8.55	30	6255	0.0044	11.76	1943
	77%			**100%**			**9.67**	**1455**

TABLE II: Comparison of **GPGPU parallelized** arGA with serial MIDACO

	arGA			MIDACO			Speed-Up	
Nr.	Optimal	Eval	Time	Optimal	Eval	Time	Eval	Time
1	30	2053	0.072	30	1803	0.0011	1.13	65
2	30	11203	0.097	30	9593	0.0068	1.17	14
3	30	24848	0.143	30	21047	0.0171	1.18	8
4	22	14401	0.080	30	2521	0.0018	5.71	44
5	25	41965	0.226	30	26367	0.0248	1.59	9
6	17	26040	0.151	30	483	0.0004	53.91	377
7	15	92698	0.370	30	101797	0.1934	0.91	2
8	19	74058	0.306	30	6255	0.0044	11.84	69
	78%			**100%**			**9.68**	**73**

When comparing the results of serial arGA with serial MIDACO a very large performance gab can be observed in Table I. While arGA struggles to solve on 5 out of 8 problems to the global optimal solution, MIDACO robustly solves each problem to the global solution in every test run an does so about 1500 times faster than arGA. When comparing the number of average function evaluation however, the picture is more moderate. On average the MIDACO algorithm is around 10 times more efficient than arGA. Only in case of problem 7, arGA requires slightly less function evaluation on average, whereas it is to note that arGA does solve this problem instance only in about 50% of the test runs, while MIDACO has a success rate of 100%.

When comparing the results of GPGPU parallelized arGA with serial MIDACO the cpu-time performance gap is not that large anymore. Nevertheless, MIDACO is 73 times faster on average. Regarding the success rate and average number of function evaluation there is no significant change to be noticed. This means, that despite the effort of reducing the algorithmic overhead of arGA by (massive) parallelization over a GPGPU, the overhead and algorithmic performance of serial MIDACO still outperforms arGA.

III. Parallelization strategy of MIDACO

In this section the parallelization strategy of MIDACO should be investigated on the same set of MINLP prob-

lems presented in Section II. The parallelization strategy of MIDACO aims on executing several problem function calls in parallel, but does not involve the parallelization of any internal parts of the ACO algorithm. The reason for this strategy is simple: Real World problems are (often) cpu-time expensive. As evolutionary algorithms do normally require a lot (often millions) of function evaluations, such algorithms are not practical for any application that is cpu-time expensive. Even if a significant speedup could be gained by reducing the algorithmic overhead of the evolutionary algorithm (see arGA in Table I and Table II), this does in no way affect the number of serial processed problem function evaluation. Therefore: For any cpu-time intensive application, the number of serial processed function evaluation dominates the overall cpu-time requirement. This diagnose is the motivation behind the parallelization strategy of MIDACO.

In order to allow the application of an evolutionary algorithm like MIDACO on a cpu-time expensive problem (where many function evaluation are required) executing several solution candidates in parallel is a promising strategy. Table III displays the average number serial processed blocks of iterates for each problem. A block denotes here the amount of parallel submitted iterates to MIDACO in one reverse communication loop (see Schlueter et al. [15]). The size of each block is determined by the parallelization factor L, which is simply the number of individual iterates per block. Like in Section II 30 test runs have been performed for every test problem. Note that in Table III the column referring to $L = 1$ represents the not-parallelized case, thus every block contains only one iterate. The results reported for this column are the same as the average function evaluation reported for MIDACO in Table I and Table II.

TABLE III: Average number of blocks, depending on L

Problem	L=1	L=2	L=10	L=100	L=1000
1	1803	1454	160	82	2
2	9593	5475	1827	519	124
3	21047	13146	4866	694	4
4	2521	1615	399	68	4
5	26367	11258	3994	578	159
6	483	372	85	17	2
7	101797	79282	16901	3566	1956
8	6255	3160	811	122	63

The results displayed in Table III show a dramatic reduction of serial processed blocks with an increasing parallelization factor L. While for the lowest possible parallelization factor ($L = 2$), a moderate reduction of about 50% can be observed for most problems, the highest investigated factor ($L = 1000$) reduces the number of average blocks even to 2 for two out of the eight problem instances.

From Table III we conclude that the parallelization strategy of MIDACO does effectively reduce the number of blocks for every problem instance. Therefore we see this parallelization strategy as promising for real world applications, where a single function evaluation is cpu-time expensive and therefore the number of serial executed evaluations is the overall time dominating factor.

IV. Real World Case Study with parallelized MIDACO

While in Section III the parallelization strategy of MIDACO was explained and theoretically investigated on a set of eight MINLP benchmarks, here it should be put to the practical test. In this section a real world application from the area of chemical engineering is considered. Namely this is the Tennessee Eastman Process (TEP), which models a complex chemical reaction that is described in Schlueter et al. [13]. Figure 1 displays the TEP flow sheet. The TEP model consist of 171 differential algebraic equations (141 algebraic equation and 30 ordinary differential equations) and was modeled in Matlab and Simulink.

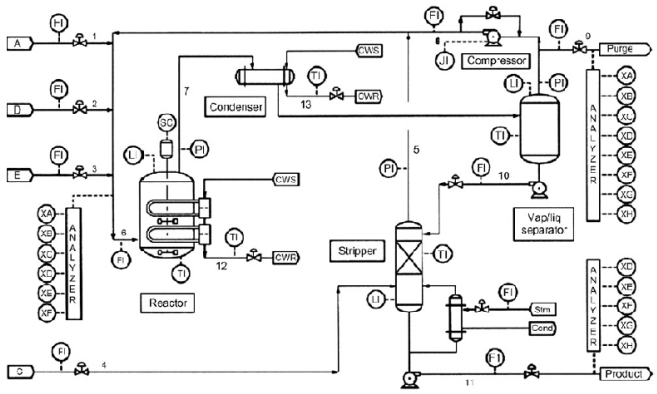

Fig. 1. Tennessee Eastman Process (TEP) plant layout

The objective of the TEP application is to minimize the annual operating cost of the chemical plant. One single model evaluation takes between 0.1 to 2.0 Seconds on an Intel I7 CPU Q820@1.73GHz. Therefore this application can be seen as a cpu-time intensive application, where a parallelization of the model evaluations might be useful.

The optimization problem that results from the TEP model is classified as MINLP and does consist of 36 continuous and 7 binary variables. Furthermore the problem is constrained due to 10 inequality and 1 equality constraint. Detailed information on this application and the MINLP optimization problem formulation can be found in Schlueter et al. [13]. The best known solution reported for the TEP model is $F(x) = 84.19$.

Two series of 10 test runs each, are considered here for the TEP application. In the first series, MIDACO is executed in serial mode, while in the second series, MIDACO is executed in parallel mode. As the above specified cpu is a quad-core processor, a parallelization factor of $L = 4$ is used. In both test series, every individual test run was either stopped if 10000 function evaluation were performed, 10000 seconds of cpu-time budget were reached or a feasible solution lower or equal 85.0 was found.

Table IV displays the results for each individual run in serial ($L = 1$) or parallel ($L = 4$) mode. In Table IV the best reached feasible objective function value is reported together with the corresponding cpu-time and number of function evaluation.

TABLE IV: Individual results of serial and parallel testruns on the TEP

	Serial (L=1)			Parallel (L=4)		
Testrun	$f(x)$	Time	Eval	$f(x)$	Time	Eval
1	84.9958	4320.83	5775	84.9620	1079.62	2916
2	98.9034	5469.58	7388	150.6043	3036.67	9056
3	149.8823	4480.24	7807	84.8861	2103.57	3988
4	84.9862	2455.59	3134	84.9018	1344.05	2288
5	84.9160	2541.74	3200	84.9041	1577.60	3316
6	84.9645	5518.65	4894	84.9069	1437.10	2608
7	84.9488	3736.28	3999	98.8107	3276.52	9812
8	84.9541	2043.18	2367	84.9133	1240.75	3124
9	86.9889	6623.58	9974	84.9830	2046.83	3804
10	150.4769	4738.92	7780	149.6352	2278.85	7408

From Table IV it can be seen that MIDACO reaches an objective function value lower or equal 85.0 in 6 out of 10 cases in serial mode. In parallel mode, MIDACO reaches an objective function value lower or equal 85.0 in 7 out of 10 cases. When comparing the average time over the successful runs in the serial and parallel case, a speed up factor of around 2.2 times can be observed. Taken into account that a relatively low parallelization factor of $L = 4$ was used, this result is coherent with the theoretical observations from Table III. While such a speed up factor does not appear to be impressive from the theoretical point of view, for a practical user it means he/she can save up to 30 minutes out of 1 hour on a personal computer (with quad-core cpu) by running MIDACO in parallel model.

V. Conclusions

An investigation of two different parallelization strategies for evolutionary algorithms for MINLP were presented. Those strategies were implemented in arGA [11] and MIDACO [16] and tested on a set of eight MINLP benchmarks. It could be shown, that a parallelization of the internal parts of the genetic algorithm arGA was not succeeding the performance of the serial implemented Ant Colony Optimization algorithm MIDACO, even though a massive parallelization by GPGPU was considered (see Table II). In contrast to this, the parallelization of the problem functions calls was theoretically investigated and practically verified on a real world application from chemical engineering. It could be shown that this parallelization strategy can theoretically provide a dramatic reduction of serial processed function calls (see Table III) if a sufficient large parallelization factor is available. The results on the real world application demonstrated the practical usefulness of this approach, where a regular calculation time of about 1 Hour could be reduced to 30 Minutes by applying a (small) parallelization factor of 4 on regular quad-core cpu computer.

References

[1] Babu, B., Angira, A.: A differential evolution approach for global optimisation of minlp problems. In: Proceedings of the Fourth Asia Pacific Conference on Simulated Evolution and Learning (SEAL 2002), Singapore, pp. 880–884 (2002)

[2] Cardoso, M.F., Salcedo, R.L., Azevedo, S.F., Barbosa, D.: A simulated annealing approach to the solution of MINLP problems. Computers Chem. Engng. 12(21), pp. 1349–1364 (1997)

[3] Deep, K., Krishna, P.S., Kansal, M.L., Mohan, C.: A real coded genetic algorithm for solving integer and mixed integer optimization problems. Appl. Math. Comput., 212(2), pp. 505–518 (2009)

[4] Duran, M. A., Grossmann, I. E.: An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs. Math. Program. 36, pp. 307-339 (1986)

[5] Floudas, C.A., Aggarwal, A., Ciric, A.R.: Global optimum search for nonconvex NLP and MINLP problems. Comput. Chem. Eng. 13(10), pp. 1117-1132 (1989)

[6] Floudas, C.A.: Nonlinear and Mixed-Integer Optimization - Fundamentals and Applications. Oxford University Press, Oxford (1995)

[7] Hock, W., Schittkowski, K.: Test Examples for Non-linear Programming Codes. Lecture Notes in Economics and Mathematical Systems, 187, Spring-Verlag, Berlin (1981)

[8] G. R. Kocis, and I. E. Grossmann: Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis. Ind. Eng. Chem. Res., 27, 1407–1421 (1988)

[9] Liang, B., Wang, J., Jiang, Y., Huang, D.: Improved Hybrid Differential Evolution-Estimation of Distribution Algorithm with Feasibility Rules for NLP/MINLP. Engineering Optimization Problems. Chin. J. Chem. Eng. 20(6), pp. 1074–1080 (2012)

[10] Powell, D., Hollingsworth, J.: A NSGA-II, web-enabled, parallel optimization framework for NLP and MINLP. Proceedings of the 9th annual conference on Genetic and evolutionary computation ,pp. 2145–2150 (2007)

[11] Munawar, A.: Redesigning Evolutionary Algorithms for Many-Core Processors Ph.D. Thesis, Graduate School of Information Science and Technology, Hokkaido University, Japan (2012)

[12] Schlüter, M., Egea, J.A., Banga, J.R.: Extended antcolony optimization for non-convex mixed integer nonlinear programming. Comput. Oper. Res. 36(7), 2217–2229 (2009)

[13] Schlueter, M., Egea, J.A., Antelo, L.T., Alonso, A.A., Banga, J.R.: An extended ant colony optimization algorithm for integrated process and control system design. Ind. Eng. Chem. 48(14), 6723–6738 (2009)

[14] Schlüter, M., Gerdts, M.: The Oracle Penalty Method. J. Global Optim. 47(2), 293–325 (2010)

[15] Schlueter, M., Gerdts, M., Rueckmann J.J.: A Numerical Study of MIDACO on 100 MINLP Benchmarks. Optimization 7(61), pp. 873–900 (2012)

[16] Schlueter, M.: Nonlinear mixed integer based Optimisation Technique for Space Applications. Ph.D. Thesis, School of Mathematics, University of Birmingham, UK (2012)

[17] Wasanapradit T., Mukdasanit N., Chaiyaratana N., Srinophakun T.: Solving mixed-integer nonlinear programming problems using improved genetic algorithms. Korean J. Chem. Eng. 28(1), 32–40 (2011)

[18] Yiqing, L., Xigang, Y., Yongjian, L.: An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints. Comp. Chem. Eng. 3(31), 153–162 (2007)

[19] Young, C.T., Zheng, Y. Yeh, C.W., Jang, S.S.: Information-guided genetic algorithm approach to the solution of MINLP problems. Ind. Eng. Chem. Res. 46, pp.1527–1537 (2007)

APPENDIX

Here the mathematical formulation of the objective function $f(x, y)$ and constraint function(s) $g_i(x, y)$ are stated for the eight considered benchmarks problems in Section II. Note that in the following, x refers to continuous optimization variables, while y refers to integer variables. For every benchmark problem, the search domain defined by lower and upper bounds, the best known solution and a literature reference is given in addition.

Example Nr. 1

$$\begin{aligned}
\text{Minimize} \quad & f(x,y) = 2x + y \\
\text{Subject to:} \quad & g_1(x,y) \geq y - 1.5x \\
& g_2(x,y) \geq 1.6 - x - y \\
\text{Bounds:} \quad & 0 \leq x \leq 1.6 \\
& 0 \leq y \leq 1 \\
\text{Solution:} \quad & f(x,y) = 2 \\
& x = 0.5 \\
& y = 1
\end{aligned}$$

This benchmark is taken from Kocis and Grossmann [8].

Example Nr. 2

$$\begin{aligned}
\text{Minimize} \quad & f(x,y) = -y + 2x_1 + x2 \\
\text{Subject to:} \quad & g_1(x,y) = x_1 - 2e^{-x_2} \\
& g_2(x,y) \geq x_1 - x_2 - y \\
\text{Bounds:} \quad & 0.5 \leq x_1 \leq 1.4 \\
& 0 \leq x_2 \leq 0.5 \\
& 0 \leq y \leq 1 \\
\text{Solution:} \quad & f(x,y) = 2.124 \\
& x_1 = 1.375183 \\
& x_2 = 0.374083 \\
& y = 1
\end{aligned}$$

This benchmark is taken from Cardoso et al. [2].

Example Nr. 3

$$\begin{aligned}
\text{Minimize} \quad & f(x,y) = -0.7y + 5(x_1 - 0.5)^2 + 0.8 \\
\text{Subject to:} \quad & g_1(x,y) \geq e^{x_1 - 0.2} + x_2 \\
& g_2(x,y) \geq -1 - x_2 - 1.1y \\
& g_3(x,y) \geq 0.2 - x_1 + 1.2y \\
\text{Bounds:} \quad & 0.2 \leq x_1 \leq 1 \\
& -2.22554 \leq x_2 \leq -1 \\
& 0 \leq y \leq 1 \\
\text{Solution:} \quad & f(x,y) = 1.07654 \\
& x_1 = 0.934169 \\
& x_2 = -2.090483 \\
& y = 1
\end{aligned}$$

This benchmark is taken from Floudas [6].

Example Nr. 4

$$\begin{aligned}
\text{Minimize} \quad & f(x,y) = 7.5y + 5.5(1-y) + 7x_1 + 6x_2 \\
& \quad + 50(1-y) \;/\; (0.8(1 - e^{-0.4x_2})) \\
& \quad + 50y \;/\; (0.9(1 - e^{-0.5x_1})) \\
\text{Subject to:} \quad & g_1(x,y) \geq 2y - 0.9(1 - e^{-0.5x_1}) \\
& g_2(x,y) \geq 2(1-y) - 0.8(1 - e^{-0.4x_2}) \\
& g_3(x,y) \geq 10y - x_1 \\
& g_4(x,y) \geq 10(1-y) - x_2 \\
\text{Bounds:} \quad & 0 \leq x_1 \leq 100 \\
& 0 \leq x_2 \leq 100 \\
& 0 \leq y \leq 1 \\
\text{Solution:} \quad & f(x,y) = 99.239635 \\
& x_1 = 3.714916 \\
& x_2 = 0 \\
& y = 1
\end{aligned}$$

This benchmark is taken from Babu and Angira [1].

Example Nr. 5

$$\begin{aligned}\text{Minimize}\quad & f(x,y) = (y_1-1)^2 + (y_2-2)^2 + (y_3-1)^2 \\ & - log(y_4+1) + (x_1-1)^2 + (x_2-2)^2 \\ & + (x_3-3)^2\end{aligned}$$

$$\begin{aligned}\text{Subject to:}\quad & g_1(x,y) \geq 5 - y_1 - y_2 - y_3 - x_1 - x_2 - x_3 \\ & g_2(x,y) \geq 5.5 - x_1^2 - x_2^2 - x_3^2 - y_3^2 \\ & g_3(x,y) \geq 1.2 - x_1 - y_1 \\ & g_4(x,y) \geq 1.8 - x_2 - y_2 \\ & g_5(x,y) \geq 2.5 - x_3 - y_3 \\ & g_6(x,y) \geq 1.2 - x_1 - y_4 \\ & g_7(x,y) \geq 1.64 - x_2 - y_2 \\ & g_8(x,y) \geq 4.25 - x_3 - y_3 \\ & g_9(x,y) \geq 4.64 - x_3 - y_2\end{aligned}$$

$$\begin{aligned}\text{Bounds:}\quad & 0 \leq x_{1,2,3} \leq 100 \\ & 0 \leq y_{1,2,3,4} \leq 1\end{aligned}$$

$$\begin{aligned}\text{Solution:}\quad & f(x,y) = 3.557466 \\ & x_1 = 0.186127 \\ & x_2 = 0.805375 \\ & x_3 = 1.908719 \\ & y_1 = 1 \\ & y_2 = 1 \\ & y_3 = 0 \\ & y_4 = 1\end{aligned}$$

This benchmark is taken from Floudas et al. [5].

Example Nr. 6

$$\begin{aligned}\text{Minimize}\quad & f(x,y) = -5.357854x_1^2 - 0.835689y_1x_3 \\ & - 37.29329y_1 + 40792.141\end{aligned}$$

$$\begin{aligned}\text{Subject to:}\quad & g_1(x,y) \geq 85.334407 + 0.0056858y_2x_3 \\ & + 0.0006262y_1x_2 - 0.0022053x_1x_3 \\ & g_2(x,y) \geq 80.51249 + 0.0071317y_2x_3 \\ & + 0.0029955y_1y_2 + 0.0021813x_1^2 - 110 \\ & g_3(x,y) \geq 9.300961 + 0.0047026x_1x_3 \\ & + 0.0012547y_1x_1 + 0.0019085x_1x_2 - 25 \\ & g_4(x,y) \geq 92 - g_1(x,y) \\ & g_5(x,y) \geq 20 - g_2(x,y) \\ & g_6(x,y) \geq 5 - g_2(x,y)\end{aligned}$$

$$\begin{aligned}\text{Bounds:}\quad & 27 \leq x_{1,2,3} \leq 45 \\ & 78 \leq y_1 \leq 102 \\ & 33 \leq y_2 \leq 45\end{aligned}$$

$$\begin{aligned}\text{Solution:}\quad & f(x,y) = -32217.4 \\ & x_1 = 27.495294 \\ & x_2 = 27 \\ & x_3 = 27.181610 \\ & y_1 = 80 \\ & y_2 = 39\end{aligned}$$

This benchmark is taken from Cardoso et al. [2].

Example Nr. 7

$$\begin{aligned}\text{Minimize}\quad & f(x,y) = 5y_1 + 8y_2 + 6y_3 + 10y_4 \\ & + 6y_5 + 7y_6 + 4y_7 + 5y_8 \\ & - 10x_1 - 15x_2 + 15x_3 + 80x_4 \\ & + 25x_5 + 35x_6 - 40x_7 + 15x_8 \\ & - 35x_9 + e^{x_1} + e^{x_2/1.2} \\ & - 65log(x_3 + x_4 + 1) \\ & - 90log(x_5 + 1) - 80log(x_6 + 1)\end{aligned}$$

$$\begin{aligned}\text{Subject to:}\quad & g_1(x,y) = 1 - y_1 - y_2 \\ & g_2(x,y) = -y_4 + y_6 + y_7 \\ & g_3(x,y) \geq 1.5log(x_5 + 1) + log(x_6 + 1) + x_8 \\ & g_4(x,y) \geq log(x_3 + x_4 + 1) \\ & g_5(x,y) \geq x_1 + x_2 - x_3 - 2x_4 - 0.8x_5 \\ & - 0.8x_6 + 0.5x_7 + x_8 + 2x_9 \\ & g_6(x,y) \geq x_1 + x_2 - 2x_4 - 0.8x_5 - 0.8x_6 \\ & + 2x_7 + x_8 + 2x_9 \\ & g_7(x,y) \geq 2x_4 + 0.8x_5 + 0.8x_6 - 2x_7 - x_8 - 2x_9 \\ & g_8(x,y) \geq 0.8Dx_5 + 0.8x_6 - x_8 \\ & g_9(x,y) \geq x_4 - x_7 - x_9 \\ & g_{10}(x,y) \geq 0.4x_5 + 0.4x_6 - 1.5x_8 \\ & g_{11}(x,y) \geq -0.16x_5 - 0.16x_6 + 1.2x_8 \\ & g_{12}(x,y) \geq -x_3 + 0.8x_4 \\ & g_{13}(x,y) \geq x_3 - 0.4x_4 \\ & g_{14}(x,y) \geq 1 - e^{x_1} + 10y_1 \\ & g_{15}(x,y) \geq 1 - e^{x_2/1.2} + 10y_2 \\ & g_{16}(x,y) \geq -x_7 + 10y_3 \\ & g_{17}(x,y) \geq -0.8x_5 - 0.8x_6 + 10y_4 \\ & g_{18}(x,y) \geq -2x_4 + 2x_7 + 2x_9 + 10y_5 \\ & g_{19}(x,y) \geq -x_5 + 10y_6 \\ & g_{20}(x,y) \geq -x_6 + 10y_7 \\ & g_{21}(x,y) \geq -x_3 - x_4 + 10y_8 \\ & g_{22}(x,y) \geq 1 - y_4 - y_5 \\ & g_{23}(x,y) \geq -y_3 + y_8\end{aligned}$$

$$\begin{aligned}
\text{Bounds:}\quad & 0 \leq x_{1,2,4,5,6,7} \leq 2 \\
& 0 \leq x_3 \leq 1 \\
& 0 \leq x_8 \leq 1 \\
& 0 \leq x_9 \leq 3 \\
& 0 \leq y_{1,2,3,4,5,6,7,8} \leq 1
\end{aligned}$$

$$\begin{aligned}
\text{Solution:}\quad & f(x,y) = 68.01 \\
& x_1 = 0.000025 \\
& x_2 = 1.985444 \\
& x_3 = 0.379430 \\
& x_4 = 0.484475 \\
& x_5 = 1.872188 \\
& x_6 = 0.000328 \\
& x_7 = 0.006863 \\
& x_8 = 0.245522 \\
& x_9 = 0.485227 \\
& y_1 = 0 \\
& y_2 = 1 \\
& y_3 = 0 \\
& y_4 = 1 \\
& y_5 = 0 \\
& y_6 = 1 \\
& y_7 = 0 \\
& y_8 = 1
\end{aligned}$$

This benchmark is taken from Duran and Grossmann [4].

Example Nr. 8

$$\begin{aligned}
\text{Minimize}\quad & f(x,y) = 5.357854x_3^2 + 0.835689x_1x_5 \\
& \quad + 37.29329x_1 - 40792.141
\end{aligned}$$

$$\begin{aligned}
\text{Subject to:}\quad & g_1(x,y) \geq 85.334407 + 0.0056858x_2x_5 \\
& \quad + 0.0006262x_1x_4 - 0.0022053x_3x_5 \\
& g_2(x,y) \geq 80.51249 + 0.0071317x_2x_5 \\
& \quad + 0.0029955x_1x_2 + 0.0021813x_3^2 - 90 \\
& g_3(x,y) \geq 9.300961 + 0.0047026x_3x_5 \\
& \quad + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 20 \\
& g_4(x,y) \geq 92 - g_1(x,y) \\
& g_5(x,y) \geq 110 - g_2(x,y) \\
& g_6(x,y) \geq 25 - g_3(x,y)
\end{aligned}$$

$$\begin{aligned}
\text{Bounds:}\quad & 78 \leq x_1 \leq 102 \\
& 33 \leq x_2 \leq 45 \\
& 27 \leq x_3 \leq 45 \\
& 27 \leq x_4 \leq 45 \\
& 27 \leq x_5 \leq 45
\end{aligned}$$

$$\begin{aligned}
\text{Solution:}\quad & f(x,y) = 30665.538669 \\
& x_1 = 78 \\
& x_2 = 33 \\
& x_3 = 30 \\
& x_4 = 45 \\
& x_5 = 36.775830
\end{aligned}$$

This benchmark is taken from Hock and Schittkowski [7].