Numerical Assessment of the Parallelization Scalability on 200 MINLP Benchmarks

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*—This contribution addresses the question if and how the impact of parallelization can influence the performance of an evolutionary algorithm on constrained mixed-integer nonlinear optimization problems. On a set of 200 MINLP benchmarks the performance of the MIDACO solver is numerically assessed with gradually increasing parallelization factor from 1 to 100. The results demonstrate that the efficiency of the algorithm can be significantly improved by parallelized function evaluation. Furthermore, the results indicate that the scale-up behaviour on the efficiency resembles a linear nature, which implies that this approach will even be promising for very large parallelization factors. The presented research is especially relevant to cpu-time consuming real-world applications, where only a low number of serial processed function evaluation can be calculated in reasonable time.**

I. Introduction

This contribution deals with the optimization of problems known as mixed-integer nonlinear programs (MINLP). The considered MINLP is stated mathematically 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 solution vector x contains the continuous decision variables and the solution vector y contains the discrete decision variables (also called *integers*). 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).

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

Optimization of MINLP problems is a young and growing field in the evolutionary computing community, see e.g. Babu and Angira [1], Cardoso [2], Costa and Oliveira [3], Deep et al. [4], Glover [6], Liang et al. [11], Mohammed [12], Munawar [13], Wasanapradit et al. [27], Young et al. [29], Yiqing et al. [28] or Yue et al. [31]. One advantage of evolutionary algorithms is their robustness towards the analytical properties of the objective and constraint functions. Therefore, in above MINLP (1) the functions $f(x,y)$ and $g(x,y)$ are considered as general black-box functions without any requirements, such as differentiability or smoothness. Another advantage of evolutionary algorithms is their capability to (greatly) benefit from parallelization, see for example Du et al. [14], Laessig and Sudholt [10], Gupta [10], Sakuray [16], Sudholt [26] or Yingyong et al [30]. One of the most popular strategies to use parallelization in evolutionary algorithms is the distributed computing of the problem function evaluations. This strategy is sometimes denoted as *co-evaluation*.

The here presented numerical study investigates the impact of a varying co-evaluation factor on the performance of an evolutionary optimization algorithm on a set of 200 MINLP benchmarks (see Schittkowski [23]), which mostly originate from the well-known GAMS MINLPlib library [9]. Here considered benchmark instances consist of up to 205 variables and 283 constraints, including up to 100 equality constraints (see the Appendix for details on the benchmark instances). The focus of this paper is to investigate and measure the efficiency of parallelized function evaluation calls on the algorithmic performance. As numerical solver, the MIDACO optimization software is used, which is based on an evolutionary algorithm especially developed for mixed-integer problems and capable of seamless parallelization of function evaluation calls.

This paper is structured as follows: In Section II a brief overview on the MIDACO algorithm is given with an emphasis on its parallelization approach. In Section III the numerical results of 100 individual test runs on the set of 200 MINLP benchmarks are illustrated and discussed. In Section IV a summary and general conclusions are presented. A comprehensive Appendix lists detailed individual information on all considered benchmarks.

II. MIDACO Optimization Algorithm

MIDACO stands for *Mixed Integer Distributed Ant Colony Optimization*. The evolutionary algorithm within MIDACO is based on the ant colony optimization metaheuristic for continuous search domains proposed by Socha and Dorigo [25] and was extended to mixed-integer domains by Schlueter et al. in [17]. For constrained optimization problems the algorithm

applies the *Oracle Penalty Method* which was introduced in Schlueter and Gerdts [18]. While the MIDACO algorithm is conceptually designed as general black-box solver, it has proven its effectiveness especially on challenging interplanetary space trajectory design problems (see Schlueter [21]), where it holds several best known record solutions on benchmarks provided by the European Space Agency [5]. It is furthermore the first algorithm that was able to successfully solve interplanetary trajectory problems formulated as mixed-integer problems, where the sequence of fly-by planets was considered as changeable integer optimization variables (see Schlueter et. al. [20].

A. MIDACO's Parallelization Approach

The parallelization approach considered here aims on distributing the problem function evaluation. This approach is sometimes referred to as co-evaluation. The MIDACO solver optimization software easily enables this kind of parallelization due to its *reverse communication* architecture. *Reverse communication* means here that the call of the objective and constraint functions happens outside and independently of the MIDACO source code.

Within a single reverse communication loop, MIDACO does accept and returns an arbitrary large number of **P** iterates x (also called "solution candidates" or "individuals") at once. Hence, those **P** iterates can be evaluated in parallel, outside and independently from the MIDACO source code. This idea of passing a **block** of **P** iterates at once within one reverse communication step to the optimization algorithm was originally introduced by the code NLPQLP by Schittkowski [24].

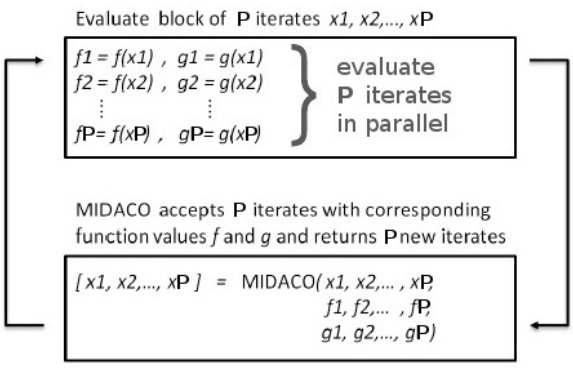

Fig. 1. Reverse communication loop with block of iterates.

Figure 1 illustrates the reverse communication loop where a **block** of **P** iterates is evaluated regarding their objective function $f(x)$ and constraints $g(x)$ and then passed to the MIDACO optimization algorithm, which then again returns a new block of **P** iterates to be evaluated.

This concept allows an independent and user controlled distributed computing of the objective and constraint function. In other words: The displayed parallelization option is valid for any language and any CPU architecture. This includes in particular multi-core PC's, PC-Clusters and GPGPU (General Purpose Graphical Processing Unit) based computation. In case of MIDACO, the parallelization factor **P** can furthermore be any arbitrary large integer value, enabling a seamless and massive parallelization. As this parallelization approach aims on distributing the function evaluation calls, it is intended for problems where the function evaluation are numerically expensive to compute, which is often the case for complex real-world applications. For further details on the parallelization approach by MIDACO, please consult [22] or [19].

III. Numerical Results

This section presents the numerical results obtained by MIDACO (5.0 beta version) on the set of 200 MINLP benchmark problems, provided by Schittkowski [23]. In total, 100 executions on the full set of 200 problem instances have been conducted. Each execution considered a different parallelization factor (see Section II-A) and a different random seed. With each execution, the parallelization factor was incrementally increased from one up to one hundred. For each individual problem out of the library a maximal number of function evaluation budget of ten million was assigned. No time limit was enforced on any run. Each individual test run on each problem was either stopped if the maximal evaluation budget was reached, or if the global[1] optimal solution was obtained.

The criteria for reaching a global optimal solution (x^*, y^*) by an approximation $(\hat{x}, \hat{y})$ is given in Equation (2).

$$\begin{aligned} f(\hat{x},\hat{y}) &\leq f(x^*,y^*) + \frac{\|f(x^*,y^*)\|}{100} \\ \|g(\hat{x},\hat{y})_{i=1,\ldots,m_e}\| &\leq 0.01 \\ g(\hat{x},\hat{y})_{i=m_{e+1},\ldots,m} &\geq -\ 0.01 \end{aligned} \tag{2}$$

Equation (2) implies that a test run was considered successful, if the approximative solution $(\hat{x}, \hat{y})$ reached by MIDACO was as close as 1% to the global optimal solution objective function value $f(x^*, y^*)$ while satisfying all constraints with a precision of at least 0.01. Note that the tolerance of 0.01 for the constraint violation is chosen here rather moderate. This is due to the relatively large number of (up to one hundred) equality constraints in several benchmark instances (see Appendix). For real-world problems, solutions with higher precision in the constraint satisfaction can normally be achieved easily with refinement runs. The lower bounds of each problem instance were used as starting point and the original bounds[2] provided by Schittkowski [23] were considered for each problem.

[1]The best known numerical $f(x, y)$ values provided in Schittkowski [23] were used as global optimal solutions throughout this study.

[2]Note that the original bounds provided in Schittkowski [23] on the problem instance are sometimes huge in the context of evolutionary computing, where the entire search space is sampled. This makes some of the instance exceptionally hard to solve with evolutionary methods.

Except for the parallelization factor, all MIDACO parameters were set to default.

Table I displays the average number of optimal and feasible solutions obtained for various test runs on the full set of 200 MINLP benchmarks. The average values displayed for each run is calculated respectively from the history of all previous runs up to the current one. The number of each run equals the parallelization factor used in such run. The abbreviations for Table I are as follows:

Run = **P**	:	Execution of MIDACO on all benchmarks
Optimal	:	Average number of global optimal solutions
Feasible	:	Average number of feasible solutions
Blocks	:	Average number of performed blocks
Evaluation	:	Average number of performed evaluation
Time	:	Cpu-time of individual test run

All numerical runs were conducted on a Desktop computer with XEON cpu with 3.47GHz clock-rate, 4GB RAM memory and six physical cores. The total time to calculate all 100 executions on the full benchmark library took 196253 seconds, which is around two and a half days. For the results presented in this study the co-evaluation of objectives and constraints was calculated on a single thread and not distributed by common parallelization schemes, such as OpenMP [8] or MPI [8]. The reason is that all benchmark function in this study take only milliseconds to compute and any actual parallelized computing scheme would introduce a computing overhead which would in fact increase overall calculation times rather than reducing them. Note that the presented results nevertheless accurately represent the factor of reduced serial processed function evaluation and that the results are therefore fully valid to estimate the performance gain in parallel executed function evaluation for cpu-time intensive real-world applications.

TABLE I: Average number of optimal and feasible solutions

Run = **P**	Optimal	Feasible	Blocks	Evaluation	Time
1	160.0	183.0	2747356	2747356	2358.5
10	158.9	180.2	284719	2847195	2144.4
20	158.1	180.1	162785	3255716	2119.4
30	157.9	180.5	107293	3218808	2042.9
40	157.7	180.4	82824	3312999	2017.9
50	157.3	180.2	67314	3365719	1945.6
60	157.1	180.1	57795	3467717	1918.6
70	156.7	179.9	51749	3622491	1875.0
80	156.5	179.7	46525	3722027	1846.8
90	156.3	179.5	39894	3590513	1804.6
100	156.0	179.4	38214	3821471	1746.0

From Table I it can be seen that MIDACO achieved in its first execution (which had a parallelization factor of one and therefore no actual parallelization) a number of 183 feasible solutions from which 160 satisfied the global optimality criteria in Equation (2). It is important to note that the number of processed **blocks** in the first run equals the number of evaluation, which where about 2.7 million (2747356). Table I illustrates that the average number of global and feasible solution gradually decreases, while the average number of function evaluation increases. This is the expected behaviour, as each run was bound to a maximal budget of 10 million function evaluation. In Table I the column of Blocks strikingly illustrates how the average number of processed blocks drastically decreases with increasing parallelization factor. While in the first run around 2.7 blocks had to be processed on average, it is only a 38,214 blocks in the last run with a parallelization factor of 100. Therefore a reduction of around $\frac{2747356}{38214} \approx 71.9$ times in the number of processed blocks, while the number of optimal solved instances dropped only around 2.5% (156 in comparison to 160).

A minor observation concerns the calculation times of each run. While the first run consumed 2358 seconds, the last run consumed only 1746 seconds. This observation can be explained by the reduced communication overhead between the MIDACO algorithm and the problem function evaluation (see Section II-A), if blocks with large number of evaluation are processed within the reverse communication loop (see Figure 1).

Figure 2 and Figure 3 display the average and individual number of optimal and feasible solutions obtained by each of the 100 runs on the full benchmark library.

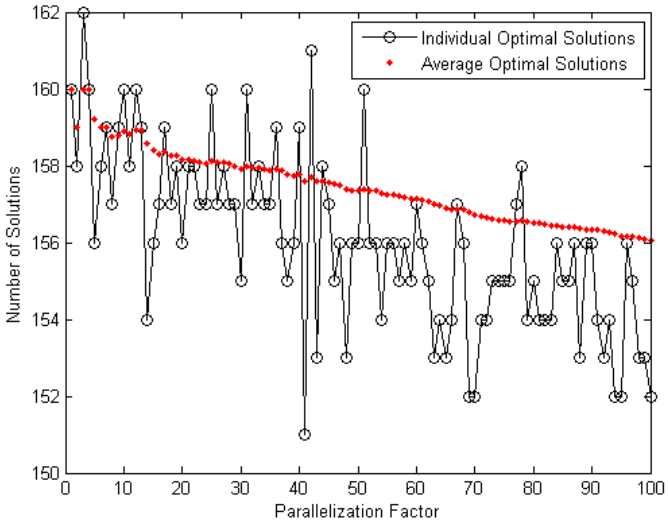

Fig. 2. Inidividual and average **optimal** solutions at run 1 to 100.

From Table I and Figure 2 and Figure 3 it can be seen, how the MIDACO algorithm benefits in drastically reducing its average number of processed blocks, while still maintain a high number of feasible and global optimal solutions.

In order to give a more sophisticated answer to the question, how the efficiency of the algorithm scales with the paralleliza-

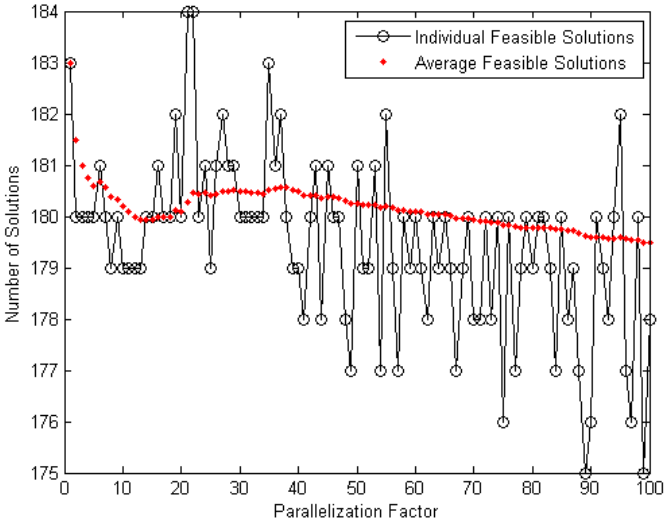

Fig. 3. Inidividual and average **feasible** solutions at run 1 to 100.

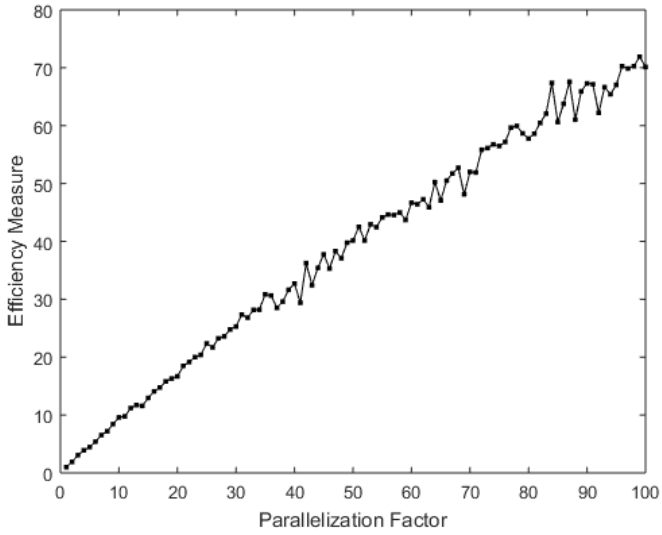

Fig. 4. Efficiency measure in regard to parallelization factor

tion factor, a new criteria for the algorithmic efficiency is now introduced. Based on the first run, which exemplifies the unparalleled performance, the "*Efficiency*" should be measured as given in Equation (3).

$$Efficiency = \frac{\#Optimal_{\ average}}{\#Optimal_{\ first\ run}} \times \frac{\#Blocks_{\ first\ run}}{\#Blocks_{\ average}} \quad (3)$$

Equation (3) measures the "*Efficiency*" based on a multiplication of a ratio of optimal obtained solutions with a ratio of required blocks. Because the number of optimal solutions is desired to be as high as possible, the average number of optimal solutions appear in the numerator of the ratio, while the number of optimal solutions from the first run appear in the denominator of the ratio. Contrary to desired number of optimal solutions, the number of blocks is desired to be as low as possible and hence the blocks required in the first run appear in the numerator, while the average number of blocks appear in the denominator. The efficiency measure given by Equation 3 can be calculated for each of the 100 runs on the full library of test problems. Figure 4 displays the efficiency measure for all 100 runs.

It is remarkable to see from Figure 4 that the scale up effect on the algorithmic efficiency resembles a nearly linear behaviour, which appears to be particularly robust for parallelization factors below 30. This behaviour indicates that parallelization will further significantly improve performance even for much larger parallelization factors. In regard to the concrete set of 200 instances, it can be seen from Figure 4, that a parallelization factor of 10 makes the MIDACO algorithm nearly 10 times more effective as in serial mode; thus, the parallelization is as most effective as possible for low parallelization factors. From Figure 4 it can also be seen, that a parallelization factor of 100 still makes the MIDACO algorithm about 71 times more effective, which was also highlighted previously in this section.

A. Additional Numerical Results

In addition to the previously presented numerical results investigating the parallelization effect on MIDACO, a separate numerical test run investigating MIDACO's capability to locate global optimal solutions is shown here. In contrast to previous numerical runs, which considered 100 executions applying a maximal function evaluation budget of 10 millions to each problem, a single (unparalleled) run on the full library with a time limit of 3 hours (10080 seconds) for each problem instance is considered here. Purpose of this additional numerical run is to evaluate the fundamental potential of MIDACO to solve even the harder instances of the test bed. Again, the lower bounds of each problem instance were used as starting point and the original bounds were considered for each problem. All MIDACO parameter were set to default.

Table II lists a summary of the results obtained by MIDACO on the full library. Note that the execution of this test run took 3.7 days of cpu-time.

TABLE II: MIDACO performance on 200 MINLP's

Number of problems in total:	200
Number of optimal solutions:	172
Number of feasible solutions:	194
Average evaluation:	7548745
Average cpu-time:	1635.3 sec
Total cpu-time:	3.7 days

From Table II it can be seen that MIDACO is able to obtain in 194 out of 200 cases a feasible solution. Out of this 194 feasible solutions, 172 solutions were globally optimal. The average number of function evaluation took around 7.5 million

which where processed in about half an hour (1635 sec) on average.

The Appendix lists the individual MIDACO results on each of the 200 MINLP instances. In regard to the number of variables, MIDACO is able to solve the largest instance in the set (see benchmark "PARALLEL" in Table IV) to global optimality in about 15 minutes. This "PARALLEL" instance considered 205 variables and 115 constraints, including 81 equality constraints. Other large instances that could be solved to global optimality include "M7" with 114 variables and 211 constraints or "MINLPHIX" with 84 variables and 92 constraints. Several instances with over a hundred varibales and/or constraints are solved to a feasible but not global optimal solution, see e.g. benchmark M6, ST_E31, RAVEM and EX1244. Note that the majority of problems containing only few variables and/or constraints are solved to global optimality within less than 0.05 seconds and most instances with ten's of variables and/or constraints are solved within few seconds or even below a second.

IV. Conclusion

A numerical assessment of the performance scalability of an evolutionary optimization algorithm on a set of 200 mixed integer nonlinear programming instances was presented. The MIDACO optimization software was chosen to represent the evolutionary algorithm as it offers mixed-integer capability combined with a seamless parallelization feature (see Section II-A). In Section III it was demonstrated, that the performance in obtaining (feasible) global optimal solutions can be significantly improved by evaluating blocks of solution candidates in parallel. Performance was understood here in a reduction of serial processed blocks, while maintaining a high number of optimal solutions. As many real-world applications are cpu-time intensive, the required number of serial processed blocks often marks the bottleneck in optimizing such applications. Hence the presented results are especially relevant to this kind of cpu-time intensive real-world applications.

Another interesting finding of this study concerns the scale-up behaviour observed. From Figure 4 in Section III it could be seen that the scale-up effect resembles a nearly linear behaviour, whereas especially for low parallelization factors (less than 30) the efficiency gain was close to its theoretical maximum. Such behaviour implies that the algorithm will further significantly benefit from even much larger parallelization factors. Given that parallelization is a growing trend in cpu-architecture, this observation is encouraging.

Acknowledgment

The authors would like to thank Prof. Klaus Schittkowski for providing the benchmark set of MINLP instances.

Appendix

This Appendix lists all 200 MINLP benchmark instances with their name and number and type of variables and constraints in Table IV. It furthermore reports the MIDACO (5.0 beta version) default performance with a maximal cpu-time budget of 3 hours (10080 seconds). For detailed information on the global optimality criteria see Section III. Note that in two benchmark cases (ST_TEST1 and WU_4) the global optimal solution lies on the lower bounds and hence equals the starting point, which implies a reported single evaluation in Table IV. Further note that in some cases where the full cpu-time budget of 3 hours (10080 seconds) was performed (and hence no global optimal solution was reached), the reported number of function evaluation exceeded 10,000,000,000 (in words: ten thousand million) and the number of evaluation is reported as asterisk in Table IV. The desire to reduce the number of (serial processed) function evalution by parallelization (see Section II-A), which is the key motivation of this contribution, becomes evident by such large values.

TABLE III: Abbreviations used in Table IV

Abbreviation	Description
Name	Name of the benchmark instance
n	Number of variables (in total)
n_i	Number of integer variables
m	Number of constraints (in total)
m_e	Number of equality constraits
Evaluation	Amount of function evaluation
Time	Amound of CPU-time (in seconds)
Status:	Solution status obtained by MIDACO
$\surd$	Status = Global optimum reached
-	Status = Feasible local solution
x	Status = Infeasible solution

TABLE IV: Individual MINLP benchmarks results (continued)

Benchmark Details					MIDACO Performance		
Name	n	n_i	m	m_e	Evaluation	Time	Status
MITP1	5	3	1	0	68	0.0	√
MITP2	5	3	7	0	34242	0.0	√
QIP1	4	4	4	0	47	0.0	√
ASAADI11	4	3	3	0	32	0.0	√
ASAADI12	4	4	3	0	173	0.0	√
ASAADI21	7	4	4	0	85	0.0	√
ASAADI22	7	7	4	0	180	0.0	√
ASAADI31	10	6	8	0	25607	0.0	√
ASAADI32	10	10	8	0	6258	0.0	√
DIRTY	25	13	10	0	2479	0.0	√
BRAAK1	7	3	2	0	547	0.0	√
BRAAK2	7	3	4	0	2938	0.0	√
BRAAK3	7	3	4	0	3666	0.0	√
DEX2	2	2	2	0	12	0.0	√
FUEL	15	3	15	6	1813806	2.0	√
WP02	2	1	2	0	9	0.0	√
NVS01	3	2	3	1	105489	0.1	√
NVS02	8	5	3	3	80510	0.1	√
NVS03	2	2	2	0	213	0.0	√
NVS04	2	2	0	0	42	0.0	√
NVS05	8	2	9	4	609879052	10800.0	-
NVS06	2	2	0	0	31	0.0	√
NVS07	3	3	2	0	199	0.0	√
NVS08	3	2	3	0	2353	0.0	√
NVS09	10	10	0	0	178	0.0	√
NVS10	2	2	2	0	106	0.0	√
NVS11	3	3	3	0	178	0.0	√
NVS12	4	4	4	0	482	0.0	√
NVS13	5	5	5	0	573	0.0	√
NVS14	8	5	3	3	27949	0.0	√
NVS15	3	3	1	0	44	0.0	√
NVS16	2	2	0	0	88	0.0	√
NVS17	7	7	7	0	1370	0.0	√
NVS18	6	6	6	0	639	0.0	√
NVS19	8	8	8	0	1668	0.0	√
NVS20	16	5	8	0	106363	0.1	√
NVS21	3	2	2	0	46938	0.0	√
NVS22	8	4	9	4	1255608863	1047.3	√
NVS23	9	9	9	0	991	0.0	√
NVS24	10	10	10	0	10205	0.0	√
GEAR	4	4	0	0	9	0.0	√
GEAR2	28	24	4	4	38616	0.1	√
GEAR2A	28	24	4	4	150078	0.2	√
GEAR3	8	4	4	4	34901	0.0	√
GEAR4	6	4	1	1	102626	0.1	√
M3	26	6	43	0	3184587	5.1	√
M6	86	30	157	0	**********	10800.0	-
M7	114	42	211	0	1730194451	10514.2	√
FLOUDAS1	5	3	5	2	2152	0.0	√
FLOUDAS2	3	1	3	0	289	0.0	√
FLOUDAS3	7	4	9	0	6540	0.0	√
FLOUDAS4	11	8	7	3	3455656	3.2	√
FLOUDAS40	11	8	7	3	6460	0.0	√
FLOUDAS5	2	2	4	0	22	0.0	√
FLOUDAS6	2	1	3	0	29	0.0	√
SPRING	17	12	8	5	2068	0.0	√

TABLE V: Individual MINLP benchmarks results (continued)

Benchmark Details					MIDACO Performance		
Name	n	n_i	m	m_e	Evaluation	Time	Status
DU_OPT5	20	13	9	0	37335	0.1	√
DU_OPT	20	13	9	0	86913	0.3	√
ST_E13	2	1	2	0	79	0.0	√
ST_E14	11	4	13	4	72356	0.1	√
ST_E15	5	3	5	2	8711	0.0	√
ST_E27	4	2	6	0	2449	0.0	√
ST_E29	11	8	7	2	857067	0.9	√
ST_E31	112	24	135	81	2058512124	10800.0	-
ST_E32	35	19	18	17	912740727	1876.8	√
ST_E35	32	7	39	15	4266383	9.6	√
ST_E36	2	1	2	1	39216	0.0	√
ST_E38	4	2	3	0	816	0.0	√
ST_E40	4	3	8	4	34269	0.0	√
ST_MIQP1	5	5	1	0	18	0.0	√
ST_MIQP2	4	4	3	0	7013	0.0	√
ST_MIQP3	2	2	1	0	778	0.0	√
ST_MIQP4	6	3	4	0	169111	0.1	√
ST_MIQP5	7	2	13	0	79462	0.1	√
ST_TEST1	5	5	1	0	1	0.0	√
ST_TEST2	6	6	2	0	46	0.0	√
ST_TEST3	13	13	10	0	9521	0.0	√
ST_TEST4	6	6	5	0	8797	0.0	√
ST_TEST5	10	10	11	0	67	0.0	√
ST_TEST6	10	10	5	0	8260	0.0	√
ST_TEST8	24	24	20	0	461832	0.7	√
TESTGR1	10	10	5	0	847	0.0	√
TESTGR3	20	20	20	0	6179	0.0	√
TESTPH4	3	3	10	0	180	0.0	√
TLN2	8	8	12	0	491	0.0	√
TLN4	24	24	24	0	60883	0.1	√
TLN5	35	35	30	0	339723	0.6	√
TLN6	48	48	36	0	2777354	6.6	√
NEJI	3	1	6	0	1641	0.0	√
TST_NAG	8	4	7	2	1168917729	10800.0	x
TLOSS	48	48	53	0	246656909	10800.0	-
TLTR	48	48	54	0	106994	0.2	√
MEANVARX	35	14	44	8	159451	0.3	√
MINLPHIX	84	20	92	30	405663248	1722.2	√
MIP_EX	5	3	7	0	9276	0.0	√
MGRID_C1	5	5	1	0	147	0.0	√
MGRID_C2	10	10	1	0	13995	0.0	√
CROP5	5	5	3	0	296	0.0	√
CROP20	20	20	3	0	18230	0.1	√
CROP50	50	50	3	0	9318	0.1	√
CROP100	100	100	3	0	348400	4.3	√
SPLITF1	12	9	9	3	12932	0.0	√
SPLITF2	24	18	15	6	21768	0.0	√
SPLITF3	24	18	15	6	207311	0.3	√
SPLITF4	24	18	15	6	40770	0.1	√
SPLITF5	24	18	15	6	270554	0.4	√
SPLITF6	24	18	15	6	16712	0.0	√
SPLITF7	36	27	21	9	69626671	140.8	√
SPLITF8	36	27	21	9	235824	0.4	√
SPLITF9	36	27	21	9	534083	1.0	√

TABLE VI: Individual MINLP benchmarks results (continued)

Benchmark Details					MIDACO Performance		
Name	n	n_i	m	m_e	Evaluation	Time	Status
ELF	54	24	38	6	3237178	9.2	√
SPECTRA2	69	30	72	9	**********	10800.0	-
WINDFAC	14	3	13	13	50720451	56.4	√
CSCHED1	76	63	22	12	**********	10800.0	-
ALAN	8	4	7	2	277922	0.2	√
PUMP	24	9	34	13	**********	10800.0	-
RAVEM	112	54	186	25	1871886913	10800.0	-
ORTEZ	87	18	74	24	**********	10800.0	-
EX1221	5	3	5	2	1995	0.0	√
EX1222	3	1	3	0	289	0.0	√
EX1223	11	4	13	4	516775	0.5	√
EX1223A	7	4	9	0	9140	0.0	√
EX1223B	7	4	9	0	1404	0.0	√
EX1224	11	8	7	2	857067	0.8	√
EX1225	8	6	10	2	27077	0.0	√
EX1226	5	3	5	1	30	0.0	√
EX1233	52	12	64	20	**********	10800.0	-
EX1243	68	16	96	24	**********	10800.0	-
EX1244	95	23	129	30	**********	10800.0	-
EX1252	39	15	43	22	857436185	10800.0	-
EX1263	92	72	55	20	**********	10800.0	-
EX1263A	24	24	35	0	458505	0.7	√
EX1264	88	68	55	20	**********	10800.0	-
EX1264A	24	24	35	0	2000871	3.0	√
EX1265	130	100	74	30	1753426609	10800.0	-
EX1265A	35	35	44	0	1018211	2.0	√
DIOPHE	4	4	1	1	22828	0.0	√
EX1266A	48	48	53	0	253634424	10800.0	-
GBD	4	3	4	0	135	0.0	√
EX3	32	8	31	17	23435814	41.0	√
EX4	36	25	30	0	227553	0.5	√
FAC1	22	6	18	10	213838	0.3	√
FAC2	66	12	33	21	3132938	9.2	√
FAC3	66	12	33	21	2965361	8.8	√
GKOCIS	11	3	8	5	34114	0.0	√
KG	9	2	9	5	242292	0.2	√
SYNTHES1	6	3	6	0	3137	0.0	√
SYNTHES2	11	5	14	1	44888	0.0	√
SYNTHES3	17	8	23	2	224927	0.3	√
PARALLEL	205	25	115	81	129831348	1159.4	√
SYNHEAT	56	12	64	20	**********	10800.0	-
SEP1	29	2	31	22	16484485	25.6	√
DAKOTA	4	2	2	0	494	0.0	√
BATCH	47	24	73	12	14533610	36.9	√
BATCHDES	19	9	19	6	216	0.0	√
ENIPLAC	141	24	189	87	1588589869	10800.0	x
PROB02	6	6	8	0	3571	0.0	√
PROB03	2	2	1	0	15	0.0	√
PROB10	2	1	2	0	447	0.0	√
NOUS1	50	2	43	41	284195041	10800.0	-
NOUS2	50	2	43	41	305474645	10800.0	-
TLS2	37	33	24	6	987927	2.1	√
TLS4	105	89	64	20	1874485649	10800.0	-
TLS5	161	136	90	30	1231187433	10800.0	-

TABLE VII: Individual MINLP benchmarks results (continued)

Benchmark Details					MIDACO Performance		
Name	n	n_i	m	m_e	Evaluation	Time	Status
OAER	9	3	7	3	28358	0.0	√
PROCSEL	10	3	7	4	526787	0.5	√
LICHOU_1	2	1	2	1	32774	0.0	√
LICHOU_2	4	2	4	0	2284	0.0	√
LICHOU_3	3	3	4	0	33	0.0	√
WU_1	32	32	0	0	178	0.0	√
WU_2	32	32	0	0	123	0.0	√
WU_3	64	64	0	0	162	0.0	√
WU_4	64	64	0	0	1	0.0	√
OPTPRLOC	30	25	30	0	11410	0.0	√
GASNET	90	10	69	48	**********	10800.0	x
TP83	5	4	6	0	538	0.0	√
TP84	5	2	6	0	4869	0.0	√
TP85	5	3	38	0	23098	0.0	√
TP87	6	2	4	4	12447	0.0	√
TP93	6	1	2	0	88573	0.1	√
FEEDTRAY	97	7	91	83	1936657925	10800.0	x
FEEDTRAY2	87	36	283	6	2047046251	10800.0	x
HILBERT20	20	20	20	20	549764	2.0	√
HILBERT50	50	50	50	50	62412742	1084.9	√
HILBERT100	100	100	100	100	107896529	6872.4	√
SLOPPY	6	6	3	0	2	0.0	√
RASTRIGIN	2	1	0	0	434	0.0	√
EMSO	6	3	4	0	948	0.0	√
TP1	2	2	0	0	17933	0.0	√
TP1A	2	2	0	0	17933	0.0	√
TP1B	2	2	0	0	17933	0.0	√
TP9	2	2	1	1	1334	0.0	√
TP10	2	2	1	0	3904	0.0	√
DEB10	182	22	129	65	1164715084	10800.0	x
IRAP1	68	68	18	0	316613	1.7	√
IRAP2	38	38	20	0	1087	0.0	√
IRAP3	40	40	21	0	3091	0.0	√
IRAP4	45	45	16	0	274366	1.1	√
IRAP5	60	60	16	0	520004	2.5	√
IRAP6	34	34	18	0	4282	0.0	√

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] Costa L., Oliveira P.: Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems. Comput Chem Eng, 25(23):257-266 (2001)

[4] 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)

[5] European Space Agency (ESA) and Advanced Concepts Team (ACT). Gtop database - global optimisation trajectory problems and solutions. Software available at http://www.esa.int/gsp/ACT/inf/op/globopt.htm, 2011.

[6] Glover, F.: Parametric tabu-search for mixed integer programs. Comput Oper Res 33(9):24492494 (2006)

[7] Gupta, S., Tan, G.: A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on GPUs. Evolutionary Computation (CEC), IEEE Congress on, Sendai, 2015, pp. 1567-1574, 2015.

[8] Quinn, J.M.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill, 2003.

[9] *GAMS MINLPlib - A collection of Mixed Integer Nonlinear Programming models*. Washington, DC, USA; software available at http://www.gamsworld.org/minlp/minlplib.htm (2016)

[10] Laessig, J., Sudholt, D.: General upper bounds on the runtime of parallel evolutionary algorithms. Evolutionary Computation, vol. 22, no. 3, pp. 405-437 (2014)

[11] 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)

[12] Mohamed, A.W.: An efficient modified differential evolution algorithm for solving constrained non-linear integer and mixed-integer global optimization problems. Int. J. Mach. Learn. & Cyber., pp.1–19 (2015)

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

[14] Du, X., Ni, Y., Yao, Z., Xiao, R.: High performance parallel evolutionary algorithm model based on MapReduce framework. Int. J. Computer Applications in Technology, Vol. 46, No. 3,m pp. 290-296 (2013)

[15] 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)

[16] Sakuray Pais, M. , Yamanaka K., Rodrigues Pinto, E.: Rigorous Experimental Performance Analysis of Parallel Evolutionary Algorithms on Multicore Platforms. In IEEE Latin America Transactions, vol. 12, no. 4, pp. 805-811, 2014.

[17] Schlueter, 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)

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

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

[20] Schlueter M., Erb S., Gerdts M., Kemble S., Rueckmann J.J.: MIDACO on MINLP Space Applications. Advances in Space Research, 51(7), 1116-1131 (2013)

[21] Schlueter M.: MIDACO Software Performance on Interplanetary Trajectory Benchmarks. Advances in Space Research, 54(4), 744-754 (2014)

[22] Schlueter, M.: MIDACO Solver - Global Optimization Software for Mixed Integer Nonlinear Programming; Software available at http://www.midaco-solver.com (2016)

[23] K. Schittkowski, *A Collection of 200 Test Problems for Nonlinear Mixed-Integer Programming in Fortran (User Guide)*, Report, Department of Computer Science, University of Bayreuth, Bayreuth, 2012

[24] K. Schittkowski, *NLPQLP - A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search (User Guide)*, Report, Department of Computer Science, University of Bayreuth, Bayreuth, 2009

[25] K. Socha and M. Dorigo, *Ant colony optimization for continuous domains*, Eur. J. Oper. Res. 85(2008), pp. 1155–1173

[26] Sudholt, D.: Parallel Evolutionary Algorithms. In Janusz Kacprzyk and Witold Pedrycz (Eds.): Handbook of Computational Intelligence, Springer, 2015.

[27] 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)

[28] 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)

[29] 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)

[30] Yingyong, Z., Yongde, Z., Qinghua, L., Jingang, J., Guangbin, Y.: Improved Multi-objective Genetic Algorithm Based on Parallel Hybrid Evolutionary Theory. International Journal of Hybrid Information Technology Vol.8, No.1, pp. 133-140 (2015)

[31] Yue T., Guan-zheng T., Shu-guang D.: Hybrid particle swarm optimization with chaotic search for solving integer and mixed integer programming problems. J. Central South Univ. 21:2731–2742 (2014)