Modified Differential Evolution in the Load Balancing Problem for the iFDAQ of the COMPASS Experiment at CERN

Ondřej Šubrt[1], Martin Bodlák[2], Matouš Jandek[1], Vladimír Jarý[1], Antonín Květoň[2], Josef Nový[1], Jan Tomsa[1] and Miroslav Virius[1]

[1]*Czech Technical University in Prague, Prague, Czech Republic*

[2]*Charles University, Prague, Czech Republic*

Keywords: Data Acquisition System, Differential Evolution, Genetic Algorithm, Load Balancing, Optimization.

Abstract: In general, state-of-the-art data acquisition systems in high energy physics experiments must satisfy high requirements in terms of reliability, efficiency and data rate capability. The paper introduces the Load Balancing (LB) problem of the intelligent, FPGA-based Data Acquisition System (iFDAQ) of the COMPASS experiment at CERN and proposes a solution based on genetic algorithms. Since the LB problem is $\mathcal{NP}$-complete, it challenges analytical and heuristic methods in finding optimal solutions in reasonable time. Differential Evolution (DE) is a type of evolutionary algorithms, which has been used in many optimization problems due to its simplicity and efficiency. Therefore, the Modified Differential Evolution (MDE) is inspired by DE and is presented in more detail. The MDE algorithm has newly-designed crossover and mutation operator and its selection mechanism is inspired by Simulated Annealing (SA). Moreover, the proposal uses an adaptive scaling factor and recombination rate affecting the exploration and exploitation of the MDE algorithm. Thus, the MDE represents a new efficient stochastic search technique for the LB problem. The proposed MDE algorithm is examined on two LB test cases and compared with other LB solution methods.

1 INTRODUCTION

The intelligent, FPGA-based Data Acquisition System (iFDAQ) (Bodlak et al., 2016; Bodlak et al., 2014) reads out data from the detectors of the COMPASS (COmmon Muon Proton Apparatus for Structure and Spectroscopy) experiment (Alexakhin et al., 2010) being a high-energy particle physics experiment with fixed-target situated on the M2 beamline of the Super Proton Synchrotron (SPS) particle accelerator at the CERN laboratory in Geneva, Switzerland.

In complex readout data systems, such as the iFDAQ, the data streams must be properly allocated in order for the load to be well-balanced in the system (Kameda et al., 1997). Thus, the necessity of solving the Load Balancing (LB) problem arises.

The paper is organized as follows. Firstly, the LB problem is introduced in Section 2. Subsection 2.1 gives the proper definition of the LB problem. Complexity of the LB problem is proved in Subsection 2.2.

Secondly, Section 3 describes the proposal of the Modified Differential Evolution (MDE) algorithm being applicable to the LB problem. All parts of the MDE algorithm are discussed in an extensive way.

Finally, numerical results are stated in Section 4 to demonstrate how the MDE approach is successful and efficient in solving the LB problem. Then, the results acquired by the MDE are compared with other LB solution methods.

2 LOAD BALANCING PROBLEM

For the iFDAQ, the most challenging task from the LB point of view is load balancing at the multiplexer (MUX) level. The optimization criterion is minimization of the difference between the output flows of the individual multiplexers. This minimization is achieved by remapping the connection of inputs to input ports of the multiplexers. Each input port establishes a connection between a data source (a detector or a data concentrator) and the MUX level. For the COMPASS experiment, it is necessary to consider flows varying from 0 B to 10 kB for each input port.

In Figure 1, a visualization of LB at the MUX level is given. There are m MUXes with p ingoing ports each. Moreover, $n \in \mathbb{N}$ flows $f_{k_1}, f_{k_2}, \ldots, f_{k_{mp}} \in$

Šubrt, O., Bodlák, M., Jandek, M., Jarý, V., Květoň, A., Nový, J., Tomsa, J. and Virius, M.
Modified Differential Evolution in the Load Balancing Problem for the iFDAQ of the COMPASS Experiment at CERN.
DOI: 10.5220/0008319102130220
In *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI 2019)*, pages 213-220
ISBN: 978-989-758-384-1

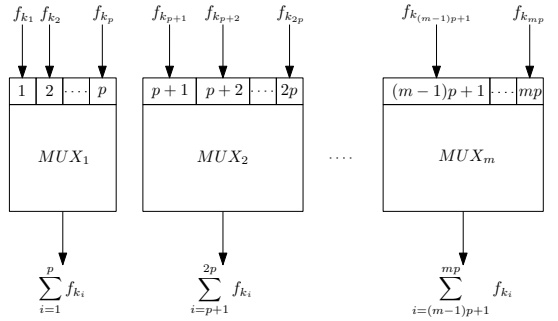

Figure 1: Visualization of LB at the MUX level.

$\mathbb{N}_0$, where $n = m \cdot p$, are shown in the figure with indices $k_1, k_2, \ldots, k_{mp} \in \{i \mid 1 \leq i \leq n\} \wedge \forall i, j : k_i \neq k_j$.

Despite the fact that each flow varies from 0 B to 10 kB in the COMPASS experiment, the domain $\mathbb{N}_0$ is used. The motivation comes from a general approach to LB. Moreover, a flow with 0 B can be either a physical connected input port sending no data or an empty input port where no data source is connected to. In brief, there are always $n = m \cdot p$ flows regardless whether all ports are used or not.

2.1 Problem Formulation

This subsection deals with a proper definition of the LB problem and preparation for discussion of the complexity of the LB problem. The Multiple Knapsack (MK) problem (Kellerer et al., 2004) is useful for the examination of the LB problem complexity as it can be shown that there exists a polynomial reduction from the MK problem to the LB problem. As MK problem is $\mathcal{NP}$-complete, this implies the LB problem is $\mathcal{NP}$-complete.

Definition 1. *Let $m \in \mathbb{N}$ denote the number of MUXes with $p \in \mathbb{N}$ ingoing ports each, i.e., $n = m \cdot p \in \mathbb{N}$ ingoing ports in total and flows $f_1, f_2, \ldots, f_n \in \mathbb{N}_0$. Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ be subsets of indices and $F = \left\lceil \sum_{i=1}^{n} f_i/m \right\rceil$ be a theoretical average flow for one MUX. The Load Balancing (LB) problem is an optimization problem such that:*

To minimize

$$\sqrt{\sum_{i=1}^{m} \left(F - \sum_{j \in \mathcal{S}_i} f_j \right)^2}, \tag{1}$$

subject to the constraints

- *each flow must be allocated*

$$\bigcup_{i=1}^{m} \mathcal{S}_i = \{i \mid i \in 1, \ldots, n\} \tag{2}$$

- *each flow must be allocated at most once*

$$\mathcal{S}_i \cap \mathcal{S}_j = \emptyset \quad \forall i, j = 1, \ldots, m \wedge i \neq j \tag{3}$$

- *each MUX has p ports*

$$|\mathcal{S}_i| = p \quad \forall i = 1, \ldots, m \tag{4}$$

Being a generalization of the well-known Knapsack problem (Kellerer et al., 2004), the MK problem represents an extension to m knapsacks.

Definition 2. *Given $n \in \mathbb{N}$ items with weights $w_1, w_2, \ldots, w_n \in \mathbb{N}$ and values $v_1, v_2, \ldots, v_n \in \mathbb{R}^+$, and $m \in \mathbb{N}$ knapsacks with capacities $W_1, W_2, \ldots, W_m \in \mathbb{N}$. The Multiple Knapsack (MK) problem is an optimization problem such that:*

To maximize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} v_j x_{ij}, \tag{5}$$

subject to the constraints

- *maximum knapsack capacity*

$$\sum_{j=1}^{n} w_j x_{ij} \leq W_i \quad \forall i = 1, \ldots, m \tag{6}$$

- *each item must be allocated at most once*

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad \forall j = 1, \ldots, n \tag{7}$$

- *assignment of item*

$$x_{ij} \in \{0, 1\} \quad \forall i = 1, \ldots, m, \quad \forall j = 1, \ldots, n \tag{8}$$

2.2 Problem Complexity

The Knapsack problem is $\mathcal{NP}$-complete (Kellerer et al., 2004). Being a generalization of the Knapsack problem, the MK problem is $\mathcal{NP}$-complete.

In order to determine the complexity of the LB problem, the decision version of the LB problem must be defined at first: *Are there subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ such that $\bigcup_{i=1}^{m} \mathcal{S}_i = \{i \mid i \in 1, \ldots, n\}$ and $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset, \forall i, j = 1, \ldots, m \wedge i \neq j$ and $|\mathcal{S}_i| = p, \forall i = 1, \ldots, m$ and $\sum_{j \in \mathcal{S}_i} f_j \leq F, \forall i = 1, \ldots, m$, where $F = \left\lceil \sum_{i=1}^{n} f_i/m \right\rceil$?*

Theorem 1. *The Load Balancing (LB) problem is $\mathcal{NP}$-complete.*

Proof. First, the LB problem is a $\mathcal{NP}$ problem. The proof are the subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ of flow indices that are chosen and the verification process is to compute $|\mathcal{S}_i| = p, \forall i = 1, \ldots, m$ and $\sum_{j \in \mathcal{S}_i} f_j \leq F, \forall i = 1, \ldots, m$, which takes polynomial time in the size of input.

Second, it will be shown there is a polynomial reduction from the MK problem to the LB problem. It suffices to show that there exists a polynomial time reduction $Q(\cdot)$ such that $Q(x)$ is a YES instance to the LB problem iff x is a YES instance to the MK problem. Suppose there are given $f_1, f_2, \ldots, f_n$, for the LB problem, consider the following MK problem: $W_i = F, V_i = p, \forall i = 1, \ldots, m$ and $w_i = f_i, v_i = 1 + f_i/h, \forall i = 1, \ldots, n$, where $h > \sum_{i=1}^{n} f_i$ and $\mathcal{K} = \bigcup_{i=1}^{m} \mathcal{K}_i \subseteq \{1, \ldots, n\}$ and $\mathcal{K}_i \cap \mathcal{K}_j = \emptyset, \forall i, j = 1, \ldots, m \wedge i \neq j$, where $\mathcal{K}_i$ represents indices of items assigned to the i-th knapsack. Here, $Q(\cdot)$ is the process converting the MK problem to the LB problem. It is clear that this process is polynomial in the input size.

If x is a YES instance for the MK problem, with the chosen sets $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_m$, let $\mathcal{R} = \{1, \ldots, n\} \setminus \bigcup_{i=1}^{m} \mathcal{K}_i$. It follows that $\sum_{j \in \mathcal{K}_i} w_j = \sum_{j \in \mathcal{K}_i} f_j \leq W_i = F, \forall i = 1, \ldots, m$ and it remains to prove there are p items in each knapsack. It follows that $\sum_{j \in \mathcal{K}_i} v_j = \sum_{j \in \mathcal{K}_i} (1 + f_j/h) \geq V_i = p, \forall i = 1, \ldots, m$ and thus, there must be at least p items in each knapsack to satisfy the inequality. Moreover, $n = m \cdot p$ implies there must be exactly p items in each knapsack and thus, $\mathcal{R} = \emptyset$. Therefore, the sets $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_m$ correspond to sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$, respectively, and x is a YES instance for the LB problem.

Conversely, if $Q(x)$ is a YES instance for the LB problem, there exists $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ such that $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset, \forall i, j = 1, \ldots, m \wedge i \neq j$ and $\bigcup_{i=1}^{m} \mathcal{S}_i = \{i \mid i \in 1, \ldots, n\}$ and $|\mathcal{S}_i| = p, \forall i = 1, \ldots, m$ and $\sum_{j \in \mathcal{S}_i} f_j \leq F, \forall i = 1, \ldots, m$. Let the MK problem consist of m knapsacks and let the i-th knapsack contain the items corresponding to indices in $\mathcal{S}_i$, and it follows that $\sum_{j \in \mathcal{K}_i} w_j = \sum_{j \in \mathcal{K}_i} f_j \leq W_i = F, \forall i = 1, \ldots, m$ and $\sum_{j \in \mathcal{K}_i} v_j = \sum_{j \in \mathcal{K}_i} (1 + f_j/h) \geq V_i = p, \forall i = 1, \ldots, m$. Therefore, $Q(x)$ is a YES instance for the MK problem.

This proves the $\mathcal{NP}$-completeness of the LB problem. □

3 MODIFIED DIFFERENTIAL EVOLUTION

Belonging to the Evolutionary Algorithms (EA) class (Eiben and Smith, 2015), a Genetic Algorithm (GA) (Affenzeller et al., 2018) is a heuristic technique inspired by the process of natural selection and evolutionary biology. It attempts to simulate evolutionary principles in order to find estimate solutions of optimization problems. These algorithms use techniques and strategies simulating processes well-known from nature – heredity, mutation, natural selection and crossover.

The main principle of the GA process is gradual production of stronger generations containing individuals representing different solutions of a problem. An individual is represented by the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$. In the optimization process, a new population is created in each generation and each individual in a population represents just one solution of a problem. As the population evolves, solutions improve.

Differential Evolution (DE) (Storn and Price, 1995; Das et al., 2016) is a stochastic, population-based search strategy developed using the same principles as GA. However, it differs significantly in the mutation step, crossover operator and the following selection mechanism. Unlike GA, a mutation of DE is applied first to generate a trial vector, which is then used within the crossover operator to produce one offspring, and the mutation step sizes are influenced by differences between the individuals of the current population.

The Modified Differential Evolution (MDE) is a heuristic algorithm based on DE and has a new mutation operation, crossover operator and selection mechanism. In this section, the newly proposed operators are described in more detail at first. Consequently, the complete MDE algorithm is presented.

3.1 Mutation Operator

The mutation operator produces a trial vector for each individual of the current population. This trial vector will then be used by the crossover operator to produce offspring.

Let the first $(i-1)$ MUXes be already allocated. In general, the mutation operator tries to mutate the i-th MUX only. The trial vector $\mathbf{u}_j(s)$ is created based on a random individual $\mathbf{x}_j^r(s)$ selected from the current population for a parent $\mathbf{x}_j(s)$ for the i-th MUX in iteration s. Firstly, all n elements from the random individual $\mathbf{x}_j^r(s)$ are copied to the trial vector $\mathbf{u}_j(s)$. Actually, the flows represented by elements with indices $1, \ldots, (i-1)p$ have already been allocated to the first $(i-1)$ MUXes. Therefore, the mutation operator does not consider elements with indices $1, \ldots, (i-1)p$ and leaves them as they are copied from the random individual $\mathbf{x}_j^r(s)$. Otherwise, the so far achieved solution

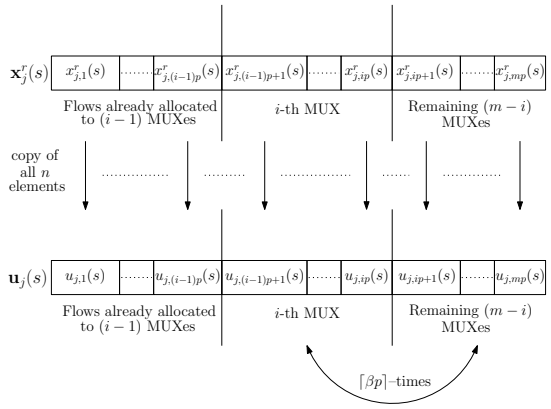

Figure 2: The mutation operator used to produce a trial vector $\mathbf{u}_j(s)$ from a random individual $\mathbf{x}^r_j(s)$ selected from current population for a parent $\mathbf{x}_j(s)$ for the i-th MUX in iteration s using $\lceil \beta p \rceil$ swaps.

would have been damaged or completely lost.

Thus, the mutation operator considers only elements with indices $(i-1)p+1,\ldots,mp$. It performs $\lceil \beta p \rceil$ swaps. In more detail, it randomly selects one element from the elements with indices $(i-1)p+1,\ldots,ip$ and one element from elements with indices $ip+1,\ldots,mp$ in each swap from $\lceil \beta p \rceil$ swaps and swaps them. To sum it up, a detailed diagram of the mutation operator is shown in Figure 2.

β is the scaling factor, controlling the amplification of the differential variation. Theoretically $\beta \in (0,\infty)$, but it is usually taken from the range [0.1, 1].

3.2 Crossover Operator

The MDE crossover operator implements a discrete recombination of the trial vector $\mathbf{u}_j(s)$ and the parent vector $\mathbf{x}_j(s)$ to produce offspring $\mathbf{x}'_j(s)$. $x_{i,j}(s)$ refers to the j-th element of the vector $\mathbf{x}_i(s)$. Elements $u_{i,j}(s)$ and $x'_{i,j}(s)$ are defined in the same way and refer to the j-th element of vectors $\mathbf{x}_i(s)$ and $\mathbf{u}_i(s)$, respectively. CR is the crossover or recombination rate in the range $[0,1]$.

The approach is similar to the mutation operator. Let the first $(i-1)$ MUXes be already allocated. In general, the crossover operator tries to cross the i-th MUX only. Firstly, all n elements from the parent $\mathbf{x}_j(s)$ are copied to the offspring $\mathbf{x}'_j(s)$. Actually, flows represented by elements with indices $1,\ldots,(i-1)p$ have already been allocated to the first $(i-1)$ MUXes. Therefore, the crossover operator does not consider elements with indices $1,\ldots,(i-1)p$ and leaves them as they are copied from the parent $\mathbf{x}_j(s)$. By analogy with the mutation operator, the so far achieved solution would be damaged or completely lost.

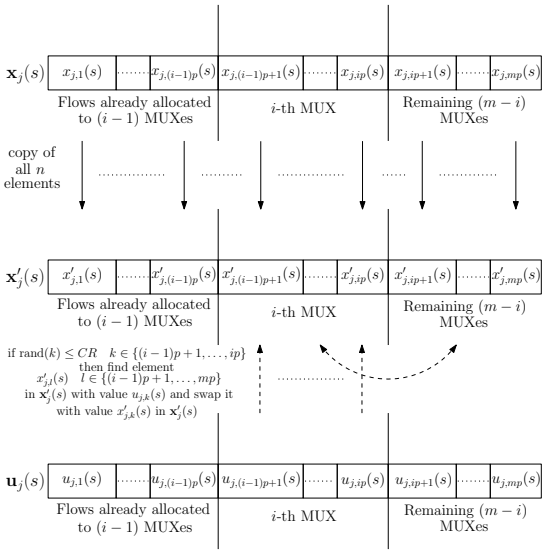

Figure 3: The crossover operator used to produce an offspring $\mathbf{x}'_j(s)$ from a parent $\mathbf{x}_j(s)$ and a trial vector $\mathbf{u}_j(s)$ for the i-th MUX in iteration s.

Thus, the crossover operator considers only the elements with indices $(i-1)p+1,\ldots,mp$. Then, for each element

$$x'_{j,(i-1)p+1}(s),\ldots,x'_{j,ip}(s), \tag{9}$$

if

$$\text{rand}(k) \le CR \quad k \in \{(i-1)p+1,\ldots,ip\}, \tag{10}$$

then find element

$$x'_{j,l}(s) \quad l \in \{(i-1)p+1,\ldots,mp\} \tag{11}$$

in $\mathbf{x}'_j(s)$ with value $u_{j,k}(s)$ and swap the values of element $x'_{j,k}(s)$ and element $x'_{j,l}(s)$ in $\mathbf{x}'_j(s)$. If the condition in Equation 10 is not satisfied, then do nothing.

In Figure 3, a diagram of the crossover operator is given. The dashed arrows represent actions being subjected to the condition in Equation 10. Once the condition is not satisfied for a given element, it does nothing and continues to the next element.

3.3 Adaptive Parameters

The scaling factor β and recombination rate CR affect the exploration and exploitation of the algorithm (Das et al., 2005). Exploration is the algorithm ability to cover and explore different areas in the feasible search space while exploitation is the ability to concentrate only on promising areas in the search space and to enhance the quality of the potential solution in the promising region. The scaling factor β controls the amplification of the differential variations. The smaller the value of β, the smaller the mutation step

sizes, and the longer it will take for the algorithm to converge. Larger values of β facilitate exploration, but may cause the algorithm to overshoot good optima. The value of β should be small enough to allow differentials to explore tight valleys, and large enough to maintain diversity. In this paper, an adaptive scaling factor is adopted to achieve a favorable compromise between exploration and exploitation. For increasing exploration, a large initial value of β is chosen. Then, it is reduced linearly along the iterations for good exploitation:

$$\beta(s) = c_1 - c_2 \frac{s}{s_{\max}}, \tag{12}$$

where $s_{\max}$ is the maximum number of iterations and c_1, c_2 are constants. In this way, the mutation operator performs a wider search in the solution space at the early stages of the evolution and at the later stages, the search is restricted around the local area.

The probability of recombination, *CR*, has a direct influence on a diversity of the MDE. The higher the probability of recombination, the more variation is introduced in a new population, thereby increasing diversity and exploration. Increasing *CR* often results in faster convergence, while decreasing *CR* increases search robustness. In the MDE, *CR* is changed along the evolution process like β as follows:

$$CR(s) = k_1 - k_2 \frac{s}{s_{\max}}, \tag{13}$$

where $s_{\max}$ is the maximum number of iterations and k_1, k_2 are constants.

3.4 Selection Mechanism

In the MDE, a probabilistic selection mechanism (Das et al., 2007) is used instead of the deterministic selection of the original DE. The selection mechanism has been inspired by Simulated Annealing (SA) (Delahaye et al., 2019). SA uses a random search strategy, which not only accepts new solutions that decrease the objective function value (assuming a minimization problem), but may also accept new solutions that rather increase the objective function value based on a predetermined probability distribution function. Exponential probability distribution function is normally used for this purpose. Based on this idea, the selection mechanism of the MDE can be described as follows:

$$\mathbf{x}_j(s+1) = \begin{cases} \mathbf{x}'_j(s) & \text{if} \quad f(\mathbf{x}'_j(s)) \le f(\mathbf{x}_j(s)) \\ \mathbf{x}'_j(s) & \text{if} \quad f(\mathbf{x}'_j(s)) > f(\mathbf{x}_j(s)) \quad \wedge \\ & \quad\; h(\mathbf{x}_j(s), \mathbf{x}'_j(s)) > \text{rand}() \\ \mathbf{x}_j(s) & \text{otherwise,} \end{cases} \tag{14}$$

$$h(\mathbf{x}_j(s), \mathbf{x}'_j(s)) = \exp\left(\frac{f(\mathbf{x}_j(s)) - f(\mathbf{x}'_j(s))}{f(\mathbf{x}_j(s))T}\right), \tag{15}$$

where T is the temperature, as defined in the SA technique. Here, the temperature T is adaptively changed in the evolution process as follows:

$$\begin{aligned} T(s+1) &= \alpha T(s) \\ T(0) &= T_0. \end{aligned} \tag{16}$$

The parameter α is the rate of reducing the temperature ($\alpha < 1$). T_0 is the initial temperature. A normalized difference between the parent and offspring objective functions has been considered in Equation 15 to eliminate the effect of different ranges of objective functions. The selection mechanism begins with a large value for T_0 and thus, many new worse solutions $\mathbf{x}_j(s)$ have a chance to be selected to increase the exploration of the MDE. However, the temperature T decreases along the iterations and so the probability of selecting worse solutions is decreased.

3.5 The MDE Algorithm

The MDE algorithm consists of the following steps, the objectives of which are described below:

Step 1 – Parameter Setup
To determine the size of the population $N_p \in \mathbb{N}$, the maximum number of iterations $s_{\max}$, constants $c_1, c_2, k_1, k_2, \alpha$ and the initial temperature T_0.

Step 2 – Initialization of Population
To initialize the individuals of the population randomly by the assignment of the input values.

Step 3 – Evaluation of Population
To evaluate the *fitness* of each individual according to the objective function in Equation 1.

Step 4 – Mutation Operation (see Subsection 3.1)
The MDE mutation operator produces a trial vector $\mathbf{u}_i(s)$ for each individual (parent) $\mathbf{x}_i(s)$ of the current population. This trial vector will then be used by the crossover operator to produce offspring.

Step 5 – Crossover (see Subsection 3.2)
The MDE crossover operator implements a discrete recombination of the trial vector $\mathbf{u}_i(s)$ and the parent vector $\mathbf{x}_i(s)$ to produce offspring $\mathbf{x}'_i(s)$.

Step 6 – Selection (see Subsection 3.4)
Either the parent $\mathbf{x}_i(s)$ or the produced offspring $\mathbf{x}'_i(s)$ survives and enters the next generation. To construct the population of the next generation, the MDE selection mechanism based on probability is used.

Step 7 – Stopping Criterion
If the stopping criterion is not satisfied, go to Step 3, else return the individual with the best *fitness* as the solution. Here, the maximum number of iterations $s_{\max}$ is selected as the stopping criterion.

Table 1: Parameters used for the MDE.

N_p	s_{max}	c_1	c_2	k_1	k_2	T_0	α
50	20,000	0.6	0.4	0.3	0.1	1	0.7

Table 2: The TC1 results using the MDE in each execution.

Ex.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
1	2.24	1,601	2.24	71,413
2	2.24	1,652	2.24	25,558
3	2.24	1,997	2.24	23,688
4	2.24	2,220	2.24	25,378
5	2.24	1,382	2.24	43,370
6	2.24	2,099	2.24	90,298
7	2.24	862	2.24	78,666
8	2.24	1,901	2.24	71,448
9	2.24	2,379	2.24	57,260
10	2.24	1,803	2.24	28,210

4 NUMERICAL RESULTS

The MDE has been implemented in C++ and Matlab (R2018a, 64-bit) on a personal computer equipped with Intel(R) Core(TM) i7-8750H CPU (@2.20 GHz, 6 Cores, 12 Threads, 9M Cache, Turbo Boost up to 4.10 GHz) and 16 GB RAM (DDR4, 2 666 MHz) memory. The MDE is examined on two test cases and numerical results are compared with other methods of solving the LB problem.

The results are investigated with respect to the error and computation time. The error is defined as the objective function of the LB problem, see Equation 1.

4.1 Test Case 1

The Test Case 1 (TC1) consists of $m = 6$ MUXes with $p = 15$ ingoing ports each and with respect to the number of MUXes, it corresponds to the iFDAQ setup used in the COMPASS Run 2016, 2017 and 2018. It considers $n = m \cdot p = 6 \cdot 15 = 90$ flows with values randomly generated in the range from 0 B to 10 kB.

The proposed MDE algorithm is partially stochastic and hence, it might produce different solutions in every execution. The parameters used for the MDE algorithm to solve the TC1 are given in Table 1.

In Table 2, the results produced in C++ and Matlab for the TC1 using the MDE in each execution are stated. The error is equal approximately to 2.24 giving a global optimum in each execution, however, the solutions might be different – multiple global optima are possible. The best TC1 flow allocation based on the MDE produced in C++ corresponding to Execution 7 is given in Figure 4 and produced in Matlab corresponding to Execution 3 is given in Figure 5. In both figures, the total flow allocated to the first MUX is 82,494 B and to the remaining five MUXes is 82,493 B each. Thus, the mutual comparison reaches $82,493/82,494 \approx 100.00\%$. The computational time is very low in C++ due to the fast memory access provided by C++ and the usage of pointers trying to avoid any copying of memory. Hence, the MDE algorithm represents a good candidate for a real-time LB solver.

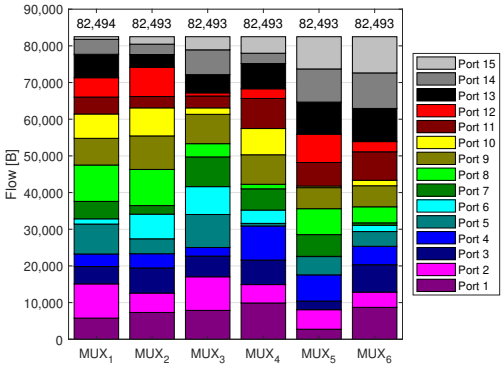

Figure 4: The best TC1 flow allocation based on the MDE produced in C++ corresponding to Execution 7.

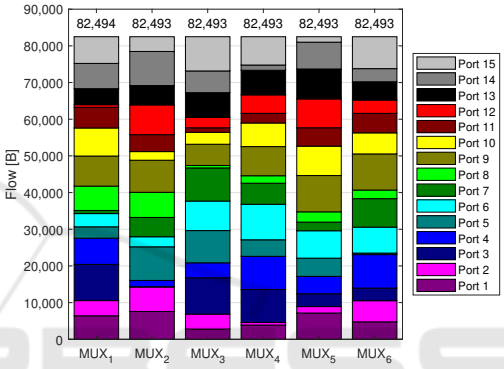

Figure 5: The best TC1 flow allocation based on the MDE produced in Matlab corresponding to Execution 3.

To demonstrate the behaviour of the MDE algorithm, the evolution process of the best TC1 flow allocation produced in Matlab corresponding to Execution 3 is shown in Figure 6. At the beginning of the evolution process, it shows a fast convergence of the proposed MDE algorithm. It reached a solution close to a global optimum after just 150 iterations. Nevertheless, there are several short-term deteriorations of the error in evolution, which are caused by selection mechanism based on SA. The selection mechanism sometimes selects individuals with a worse *fitness* value. They have a chance to show their potential to produce a new population. However, this feature weakens at the end of the evolution process.

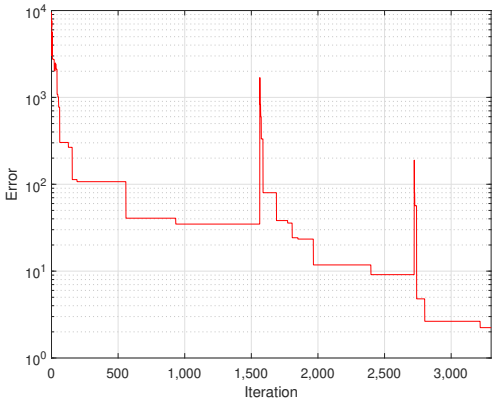

Figure 6: The evolution process of the best TC1 flow allocation based on the MDE produced in Matlab corresponding to Execution 3.

Table 3: Comparison of LB solution methods based on the best flow allocation for the TC1.

Met.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
DP	2.24	17,529	2.24	16,521
GH	243.89	47	243.89	1
ILP	21.19	17,127	23.81	1,465
MDE	2.24	862	2.24	23,688
RL	10.44	32,404	10.44	117,106

Finally, a global optimum is reached after 3,400 iterations and the error is equal approximately to 2.24.

The TC1 results produced by the MDE are compared with other LB solution methods – Dynamic Programming (DP), Greedy Heuristic (GH), Integer Linear Programming (ILP) and Reinforcement Learning (RL) – in Table 3. In sum, the MDE and GH, both reaching a global optimum, can be used for real-time LB due to the low computational time.

Table 4: The TC2 results using the MDE in each execution.

Ex.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
1	15.30	6,050	2.00	87,064
2	2.00	2,507	2.00	67,601
3	2.00	1,911	2.00	104,993
4	2.00	3,266	2.00	61,987
5	2.00	2,358	2.00	94,706
6	2.00	2,659	2.00	80,280
7	2.00	5,398	4.24	151,224
8	2.00	3,153	2.00	108,325
9	2.00	2,718	2.00	37,530
10	2.00	5,100	2.00	82,067

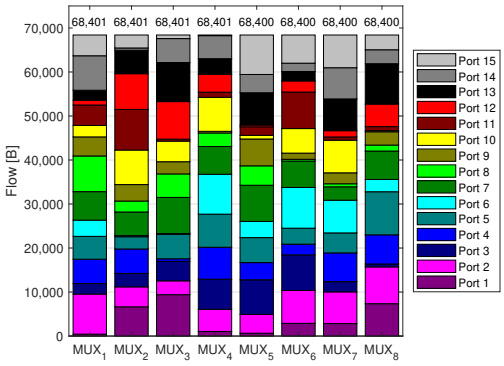

Figure 7: The best TC2 flow allocation based on the MDE produced in C++ corresponding to Execution 3.

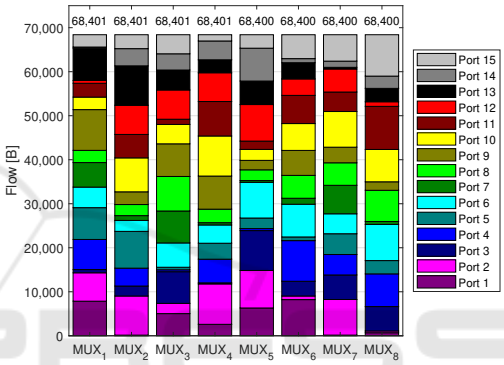

Figure 8: The best TC2 flow allocation based on the MDE produced in Matlab corresponding to Execution 9.

4.2 Test Case 2

The Test Case 2 (TC2) consists of $m = 8$ MUXes with $p = 15$ ingoing ports each and thus, it corresponds to the iFDAQ full setup. However, the iFDAQ full setup has never been in operation for the COMPASS experiment since it was not required by any physics program. It considers $n = m \cdot p = 8 \cdot 15 = 120$ flows with values randomly generated in the range from 0 B to 10 kB. The parameters used for the MDE algorithm to solve the TC2 are the same as for TC1, see Table 1.

In Table 4, the results produced in C++ and Matlab for the TC2 using the MDE in each execution are stated. The error is almost always equal to 2.00 giving a global optimum in each execution. The best TC2 flow allocation based on the MDE produced in C++ corresponding to Execution 3 is given in Figure 7 and produced in Matlab corresponding to Execution 9 is given in Figure 8. In both figures, the total flow allocated to the first four MUXes is 68,401 B each and to the remaining four MUXes is 68,400 B each. Thus, the mutual comparison reaches $68,400/68,401 \approx 100.00\%$.

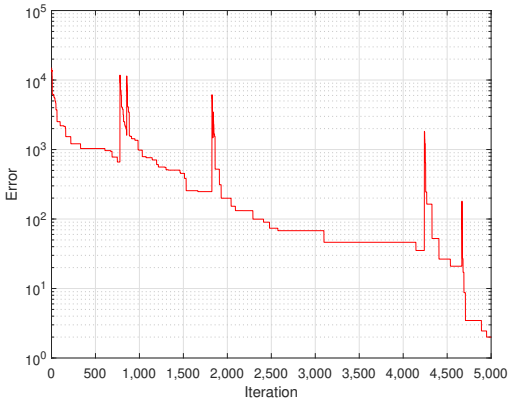

Figure 9: The evolution process of the best TC2 flow allocation based on the MDE produced in Matlab corresponding to Execution 9.

Table 5: Comparison of LB solution methods based on the best flow allocation for the TC2.

Met.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
DP	2.00	24,873	2.00	23,873
GH	224.22	63	224.22	1
ILP	194.43	49,927	134.61	95,251
MDE	2.00	1,911	2.00	37,530
RL	2.83	75,882	2.45	204,824

The evolution process of the best TC2 flow allocation based on the MDE produced in Matlab corresponding to Execution 9 can be seen in Figure 9. A global optimum is reached after 4,900 iterations and the error is equal to 2.00. Finally, the TC2 results produced by the MDE are compared with results acquired by DP, GH, ILP and RL in Table 5.

5 CONCLUSION

The paper has introduced the LB problem of the iFDAQ of the COMPASS experiment at CERN. $\mathcal{NP}$-completeness of the LB problem makes optimization more challenging. The proposed MDE has a new crossover and mutation operator and its selection mechanism is inspired by SA. Results have shown the MDE matches requirements in terms of the best error and ability to find a global optimum. Thus, the MDE represents a solver of the long-term LB setup, where no frequent changes in the flows are expected.

Since 2019, a crosspoint switch connecting all involved links in the iFDAQ provides a fully programmable system topology making the iFDAQ reconfigurable on-the-fly and replaces the fixed point-to-point connections. Thus, the crosspoint switch will analyze flows and automatically assign them to input ports of MUXes in order to equally distribute load over all MUXes. The low computational time of the MDE opens up a perspective for real-time LB.

ACKNOWLEDGEMENTS

This research has been supported by OP VVV, Research Center for Informatics, CZ.02.1.01/0.0/0.0/16_019/0000765.

REFERENCES

Affenzeller, M., Wagner, S., Winkler, S., and Beham, A. (2018). *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Taylor & Francis Ltd, first edition.

Alexakhin, V. Y. et al. (2010). *COMPASS-II Proposal*. The COMPASS Collaboration. CERN-SPSC-2010-014, SPSC-P-340.

Bodlak, M. et al. (2014). FPGA based data acquisition system for COMPASS experiment. *Journal of Physics: Conference Series*, 513(1):012029.

Bodlak, M. et al. (2016). Development of new data acquisition system for COMPASS experiment. *Nuclear and Particle Physics Proceedings*, 273(Supplement C):976–981. 37th International Conference on High Energy Physics (ICHEP).

Das, S., Konar, A., and Chakraborty, U. K. (2005). Two Improved Differential Evolution Schemes for Faster Global Search. In *GECCO 2005 – Genetic and Evolutionary Computation Conference*, pages 991–998.

Das, S., Konar, A., and Chakraborty, U. K. (2007). Annealed Differential Evolution. In *2007 IEEE Congress on Evolutionary Computation*, pages 1926–1933.

Das, S., Mullick, S. S., and Suganthan, P. N. (2016). Recent Advances in Differential Evolution -– An Updated Survey. *Swarm and Evolutionary Computation*, 27:1–30.

Delahaye, D., Chaimatanan, S., and Mongeau, M. (2019). *Simulated Annealing: From Basics to Applications*, pages 1–35. Springer International Publishing.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer-Verlag Berlin Heidelberg, second edition.

Kameda, H., Li, J., Kim, C., and Zhang, Y. (1997). *Optimal Load Balancing in Distributed Computer Systems*. Springer-Verlag London, first edition.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer-Verlag Berlin Heidelberg, Berlin, Germany, first edition. ISBN 978-3-540-24777-7.

Storn, R. and Price, K. (1995). Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. *Journal of Global Optimization*, 23.