Accelerated Unmixing: A Heterogeneous Computing Approach to fastICA for Multi-Channel EEG Analysis

Sophia L. Ramirez[1*] Nikolai Petrov[2†] Wei-Han Zhang[3‡]
Aadarsh Patel[4§] Elena Kostadinova[5¶]

April 9, 2025

Abstract

High-density electroencephalographic (EEG) systems are utilized in the study of the human brain and its underlying behaviours. However, working with the EEG data requires a well-cleaned signal, which is often obtained using independent component analysis (ICA) methods. The longer the calculation time for these types of algorithms is, the more data is obtained. This paper presents a hybrid implementation of the fastICA algorithm that uses parallel programming techniques (libraries and extensions of the Intel processors and CUDA programming), which results in a significant acceleration of execution time on selected architectures.

Keywords— ICA, EEG, BLAS, MKL, OpenMP, Intel Cilk Plus, CUDA

1 Introduction

EEG systems are commonly used in research on brain function as well as to investigate the neural basis of behaviour or cognition. Unlike the other brain imaging techniques such as functional magnetic resonance imaging (fMRI) or positron emission tomography (PET), which have a relatively low spatial resolution, high-density EEG systems can provide an accurate view of brain activity, with an electrode resolution of up to several hundred sensors. These systems have a wide range of applications in such fields as neuroscience, psychology, and medicine. However, thorough cleaning of the signal of artifacts (parts of the signal that do not originate from the brain) is crucial for its further analysis, however, this process can be time-consuming [12, 2, 21]. Additionally, manufacturers of EEG systems do not always meet the expectations of researchers and

*slramirez@ucsd.edu
†n.petrov@kurchatov.ru
‡whzhang@tsinghua.edu.cn
§aadarsh.patel@intel.com
¶e.kostadinova@tu-sofia.bg

the methods available in their tools are not satisfactory enough. The most frequently used methods of signal purification are ICA-type algorithms [8, 11, 19, 36] and they are most often available in other external programs, not adapted to larger batches of data. ICA is a powerful tool for analyzing and interpreting complex data sets and has a wide range of applications in such fields as machine learning or data mining. However, due to the iterative nature of ICA-type algorithms, their computation time can be daunting. For example, for a 256-electrode EGI system with 1000 Hz sampling [30], it can take several hours to process a 10-minute study. This paper describes an implementation that accelerates these calculations by exploiting the capabilities of multi-core architectures. Initially, the implementation was based on parallel calculations on CPU cores using the Intel processors, and over time it was enriched with matrix calculations using the CUBLAS library, which is available through programming in CUDA for the NVIDIA graphics cards [17, 15, 16, 18]. The paper presents improvement that transfers more calculations to the graphics card, thus reducing the calculation time even further. The calculation time obtained in this way was compared with the implementation time without the CUDA support (see Fig. 1).

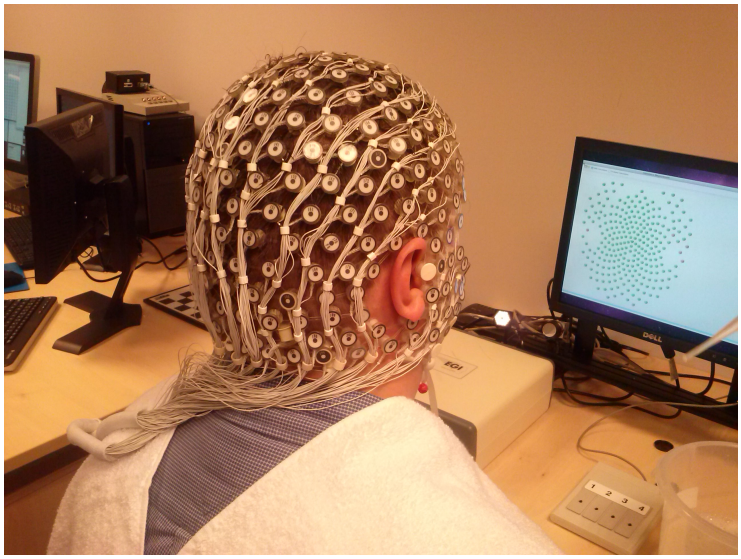


Figure 1: 345-electrode cap in EEG environment

2 Related Work

The fastICA algorithm [19] has become a cornerstone technique for blind source separation in EEG signal processing due to its computational efficiency and reliable performance. Since its introduction, numerous studies have explored various optimization strategies to enhance its performance, particularly for high-density EEG data processing.

2.1 Independent Component Analysis for EEG

Independent Component Analysis (ICA) has been widely adopted for EEG signal processing, particularly for artifact removal and source separation [8, 11]. The application of ICA in biomedical signal processing has been extensively studied [36], with various implementations tailored for EEG data analysis. Hyväarinen and Oja [20] provided a comprehensive overview of ICA algorithms and applications, establishing a foundation for subsequent optimization efforts. In the context of high-density EEG systems, which can record from up to 256 electrodes simultaneously [30], the computational demands of ICA become substantial, necessitating efficient implementations.

Wojcik et al. [38, 40, 39] demonstrated the clinical relevance of advanced EEG analysis techniques for psychiatric disorders, highlighting the need for computationally efficient methods that can process high-dimensional data in reasonable time frames. These applications have driven research toward optimizing ICA algorithms for both accuracy and computational efficiency.

2.2 CPU-Based Optimizations

Several studies have focused on CPU-based optimizations for ICA algorithms. Gajos-Balińska et al. [16, 17] investigated parallel implementations of ICA for EEG signals, demonstrating significant performance improvements through effective parallelization strategies on multi-core architectures. Their work on performance optimization of ICA algorithms for EEG data established benchmark performance metrics for CPU implementations.

The utilization of Intel's advanced vector extensions (AVX) has been explored for various computational tasks, with Rahman [33] providing a comprehensive guide to Intel Xeon Phi architecture and tools for application developers. This architecture offers substantial potential for accelerating matrix operations common in ICA algorithms. Similarly, Szustak et al. [35, 34] demonstrated the effectiveness of performance-portable parallel programming across shared-memory platforms with modern Intel processors, establishing approaches that could be adapted for ICA implementations.

2.3 Hybrid CPU-GPU Implementations

The potential of hybrid CPU-GPU architectures for accelerating signal processing algorithms has been increasingly recognized. Gajos-Balińska et al. [18] specifically addressed the cooperation of CUDA and Intel multi-core architecture in the independent component analysis algorithm for EEG data, representing one of the most relevant precursors to our work. Their study demonstrated the feasibility of distributing ICA computations across heterogeneous computing resources but did not fully optimize the workload distribution for high-density EEG data.

Lastovetsky et al. [23] presented a model-based optimization approach for kernel execution on Intel Xeon Phi through load imbalancing, which provided valuable insights into workload distribution strategies for heterogeneous computing environments. While not specifically focused on ICA or EEG processing, their approach informed the workload distribution strategy developed in our work.

3 Our Methodology

In this work, we created a data set and proposed a system for detecting NLI in Arabic sentences where the target labels were entailment, contradiction, and neutral (no semantic relationship). Our system consists of three main parts: text pre-processing (cleaning, tokenization, stemming), feature extraction (contradiction feature vector and language model vectors), and the machine-learning model. Figure 2 shows our experimental schema. We will discuss each step in detail in the following subsections.

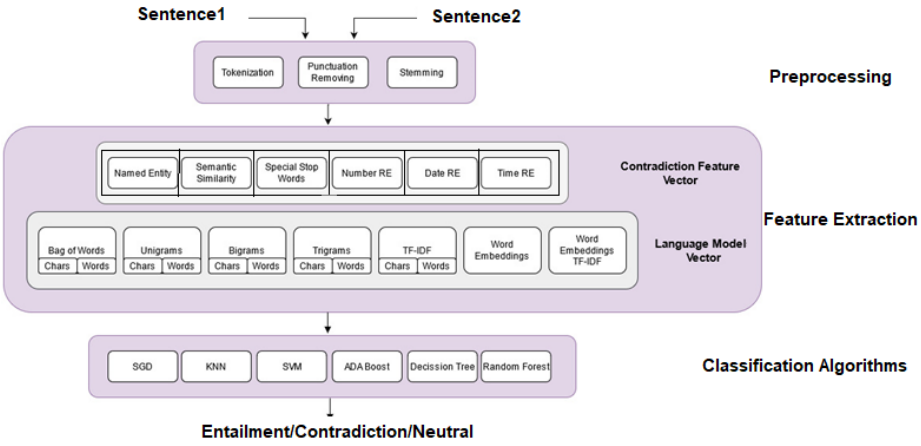


Figure 2: Our experimental schema

3.1 Our Data Set

To the best of our knowledge, there is no available Arabic three-way natural language inference (NLI) data set. In order to build our data set, we started by translating two English RTE data sets: the SICK data set [26] (which was used in SemEval_2014_Task1),[1] and the PHEME data set. The SICK data set consisted of 10,000 English sentence pairs, each annotated for relatedness in meaning and entailment relationship, while the PHEME data set contained 5400 RTE annotated pairs from social media. We named these automatically translated Arabic data sets Ar_SICK and Ar_PHEME, respectively. After automatically translating the two data sets, we selected a subset of the annotated pairs and manually corrected their translations. We augmented this subset with manually translated/annotated pairs from pre-existing sources. Our final Arabic natural language inference (NLI) data set[2] (ArNLI) contained 6366 pairs that were divided as 1932 entailment, 1073 contradiction, and 3361 neutral. The data set was collected as follows:

- **5948 pairs** of AR_SICK data set sentences that were semi-automatically translated and corrected (1714 entailment, 895 contradiction, and 3339 neutral pairs);

[1] semeval2014 task1 2014 `https://alt.qcri.org/semeval2014/task1`
[2] ArNLI `https://github.com/Khloud-AL/ArNLI`

• **312 pairs** of ArbTEDS corpus[3] from which we had to re-annotate its sentence classes (entails, not-entails) into the three-way RTE classes that were considered in this study (194 entailment, 113 contradiction, and 5 neutral pairs);

• **35 pairs** of Stanford real-life contradiction corpus [10], which was manually translated (0 entailment, 35 contradictions, and 0 neutral pairs);

• **71 pairs** of manually annotated sentences (collected from online websites teaching Arabic contradiction, poems, idioms, and paraphrased pairs of Ar_PHEME data set) with 24 entailments, 30 contradictions, and 17 neutral pairs.

The key statistics of our created data set (ArNLI) are shown in Table 1.

Table 1: Key statistics of ArNLI data set

Data Size	
Training pairs	5092
Testing pairs	1274
Avg. Sentence Length in tokens	
Hypothesis	6.623
Premise	7.246
Max. Sentence Length in tokens	
Hypothesis	26
Premise	57

3.2 Text Preprocessing

In this step, we first tokenized the sentences and removed all of the punctuation marks. To extract the morphological units, we used Snowball Stemmer (which is also known as the Porter2 stemming algorithm) Table 2 presents examples of each step in preprocessing stage.

3.3 Feature Extraction

In our proposed model, we used different types of features: named entity features, WordNet::Similarity features, special stopword feature, and number, date, and time features. We used different language models such as TFIDF, n-grams, and word embeddings.

Table 3 demonstrates the mean and standard deviation statistical results of 100 independent simulations with 1000 iterations each for all of the algorithms on the Appendicitis data set. The results showed that the proposed algorithm provided the best mean result across all performance metrics as compared to the comparative algorithms. Moreover, the statistical test also confirmed the superior statistical performance of our proposed algorithm when compared to the JAYA, PSO, SCA, and k-means algorithms with respect to all of the performance metrics.

[3]Arabic textual entailment data set `http://www.cs.man.ac.uk/~ramsay/ArabicTE/`

Table 2: Examples of output of each step in preprocessing stage

Stage	Sentence 1	Sentence 2
	عملنا في هذا البحث على فهم علاقات الاستدلال و استخراجها بين الجمل في جميع اللغات، وليس فقط اللغة العربية ..	عملنا في هذا البحث على اكتشاف علاقات الاستدلال و التناقضات بين الجمل في اللغة العربية فقط، لم نعمل على اكتشافها في باقي اللغات !
Tokenization	(['..', 'عملنا', 'في', 'هذا', 'البحث', 'على', 'فهم', 'علاقات', 'الاستدلال', 'و', 'استخراجها', 'بين', 'الجمل', 'في', 'جميع', 'اللغات،', 'وليس', 'فقط', 'اللغة', 'العربية'])	(['عملنا', 'في', 'هذا', 'البحث', 'على', 'اكتشاف', 'علاقات', 'الاستدلال', 'و', 'التناقضات', 'بين', 'الجمل', 'في', 'اللغة', 'العربية', 'فقط،', 'لم', 'نعمل', 'على', 'اكتشافها', 'في', 'باقي', 'اللغات', '!'])
Punctuation Removal	['عملنا', 'في', 'هذا', 'البحث', 'على', 'فهم', 'علاقات', 'الاستدلال', 'و', 'استخراجها', 'بين', 'الجمل', 'في', 'جميع', 'اللغات،', 'وليس', 'فقط', 'اللغة', 'العربية']	['عملنا', 'في', 'هذا', 'البحث', 'على', 'اكتشاف', 'علاقات', 'الاستدلال', 'و', 'التناقضات', 'بين', 'الجمل', 'في', 'اللغة', 'العربية', 'فقط،', 'لم', 'نعمل', 'على', 'اكتشافها', 'في', 'باقي', 'اللغات']
Snowball Stemmer	['عمل', 'في', 'هذا', 'بحث', 'على', 'فهم', 'علاق', 'استدلال', 'و', 'استخراج', 'بين', 'جمل', 'في', 'جميع', 'اللغ', 'ليس', 'فقط', 'اللغ', 'عرب']	['عمل', 'في', 'هذا', 'بحث', 'على', 'اكتشاف', 'علاق', 'استدلال', 'و', 'تناقض', 'بين', 'جمل', 'في', 'اللغ', 'عرب', 'فقط', 'لم', 'نعمل', 'على', 'اكتشاف', 'في', 'باق', 'اللغ']

Table 3: Performance of different algorithms considering accuracy, specificity, F-score, and MCC metrics on Appendicitis data set

	Accuracy Mean ± Std. Dev.	**Specificity Mean ± Std. Dev.**	**F-score Mean ± Std. Dev.**	**MCC Mean ± Std. Dev.**
Proposed	**84.1198 ± 0.3401**[a]	**0.7480 ± 0.0266**[a]	**0.8545 ± 0.0040**[a]	**0.6952 ± 0.0068**[a]
GWO	83.0049 ± 0.2638[b]	0.7229 ± 0.0238[b]	0.8465 ± 0.0019[b]	0.6764 ± 0.0035[b]
JAYA	79.1217 ± 0.6563[e]	0.7237 ± 0.1132[b]	0.8027 ± 0.0183[e]	0.6027 ± 0.0112[e]
PSO	80.2758 ± 3.5350[d]	0.6924 ± 0.0986[c]	0.8227 ± 0.0270[c]	0.6269 ± 0.0563[d]
SCA	81.3237 ± 1.1710[c]	0.7340 ± 0.0643[ab]	0.8266 ± 0.0129[c]	0.6391 ± 0.0197[c]
K-means	77.6500 ± 0.2781[f]	0.00399 ± 0.0004[d]	0.8082 ± 0.0028[d]	0.5861 ± 0.0071[f]

4 Materials and Methods

4.1 Problem Statement

A complete graph K_n includes a set of n vertex $V = \{v_1, v_2, ..., v_n\}$ and a distance matrix $C = \{c(v_i, v_j) \mid i, j = 1, 2, ..., n\}$ ($c(v_i, v_j)$ that is the cost to travel from vertex v_i to v_j). A resource matrix $RM = \{r(v_i, v_j)\}$ shows the required resource consumption to travel from vertex v_i to v_j. Let $R = (1, 2, ..., k)$ be a set of k vehicles. All vehicles start at a depot $s = v_1$. Let RM_{max} be the maximum total resources of all vehicles. A route $T = (R_1, ..., R_l, ..., R_k)$ consists of a set of routes. Each route

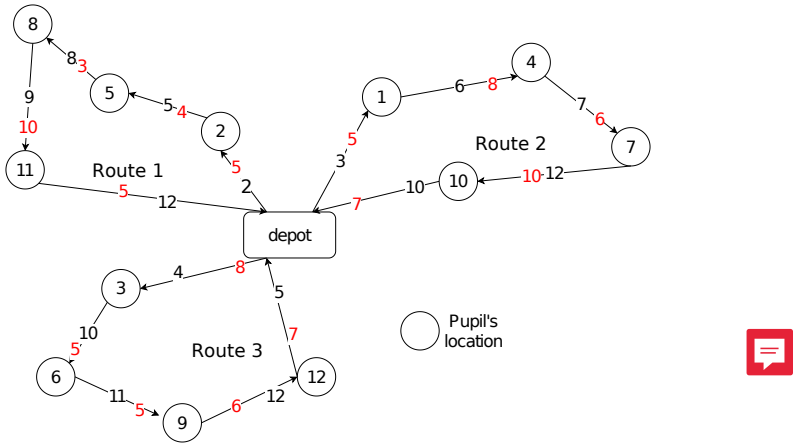


Figure 3: Example of how SBDP-RC is represented in drawing

$R_l = (v_1, ..., v_h, ..., v_m, v_{m+1} = v_1)$ is created by vehicle $l - th$. The waiting time of v_h $(1 < h \leq m)$ on R_l is the cost of the path from v_1 to v_h:

$$l(P(v_1, v_h)) = \sum_{i=1}^{h-1} c(v_i, v_{i+1}). \tag{1}$$

Let $W(R_l)$ be the total of the waiting times of R_l, and the resource consumption of route R_l (LR) is the total of the resource consumption on its edges.

$$W(R_l) = \sum_{h=2}^{m+1} l(P(v_1, v_h)); \tag{2}$$

$$LR(R_l) = \sum_{i=1}^{m} r(v_i, v_{i+1}). \tag{3}$$

The aim is as follows:

$$W(T) = \sum_{l=1}^{k} W(R_l) \rightarrow \min. \tag{4}$$

The resource consumption of each vehicle must satisfy the following:

$$\sum_{l=1}^{k} LR(R_l) \leq RM_{max}. \tag{5}$$

SBDP-RC requires a solution that begins at v_1 and visits each vertex exactly once such that the waiting times of the route are minimized. In this problem, we are interested in a Hamiltonian cycle; this means that the deliverymen return to the vertex from which they began their routes. Consider the example of the small graph that is shown in Figure 3. Assume that we have complete graph $K_{12} = 0 \cup \{1, 2, 3, ..., 12\}$. All vehicles start at the main depot (vertex 0), and each pupil's location corresponds to a vertex in the graph. The cost values to travel between two vertices are highlighted

in black, while the resource consumption values are highlighted in red. We have route $T = (R1 = (v_0, v_2, v_5, v_8, v_{11}, v_0), R2 = (v_0, v_1, v_4, v_7, v_{10}, v_0), R3 = (v_0, v_3, v_6, v_9, v_{12}, v_0))$. Assume that the value of RM_{max} is 100; the waiting times for each route are calculated as follows:

$$\begin{aligned} W(R_1) &= c(v_0, v_2) + c(v_0, v_2) + c(v_2, v_5) + c(v_0, v_2) \\ &\quad + c(v_2, v_5) + c(v_5, v_8) + c(v_0, v_2) \\ &\quad + c(v_2, v_5) + c(v_5, v_8) + c(v_8, v_{11}) \\ &\quad + c(v_5, v_8) + c(v_0, v_2) + c(v_2, v_5) \\ &\quad + c(v_5, v_8) + c(v_8, v_{11}) + c(v_{11}, v_0) \\ &\quad = 2 + (2 + 5) + (2 + 5 + 8) \\ &\quad + (2 + 5 + 8 + 9) + (2 + 5 + 8 + 9 + 12) \\ &\quad = 84. \end{aligned}$$

$$\begin{aligned} W(R_2) &= c(v_0, v_1) + c(v_0, v_1) + c(v_1, v_4) + c(v_0, v_1) \\ &\quad + c(v_1, v_4) + c(v_4, v_7) + c(v_0, v_1) \\ &\quad + c(v_1, v_4) + c(v_4, v_7) + c(v_7, v_{10}) \\ &\quad + c(v_0, v_1) + c(v_1, v_4) + c(v_4, v_7) \\ &\quad + c(v_7, v_{10}) + c(v_{10}, v_0) \\ &\quad = 3 + (3 + 6) + (3 + 6 + 7) \\ &\quad + (3 + 6 + 7 + 12) + (3 + 6 + 7 + 12 + 10) \\ &\quad = 94. \end{aligned}$$

$$\begin{aligned} W(R_3) &= c(v_0, v_3) + c(v_0, v_3) + c(v_3, v_6) \\ &\quad + c(v_0, v_3) + c(v_3, v_6) + c(v_6, v_9) \\ &\quad + c(v_0, v_3) + c(v_3, v_6) + c(v_6, v_9) \\ &\quad + c(v_9, v_{12}) + c(v_0, v_3) + c(v_3, v_6) \\ &\quad + c(v_6, v_9) + c(v_9, v_{12}) + c(v_{12}, v_0) \\ &\quad = 4 + (4 + 10) + (4 + 10 + 11) \\ &\quad + (4 + 10 + 11 + 12) \\ &\quad + (4 + 10 + 11 + 12 + 5) = 122. \end{aligned}$$

The waiting times for the route are as follows:

$$W(T) = 84 + 94 + 122 = 300.$$

The resource consumption of each route is as follows:

$$\begin{aligned} LR(R_1) &= r(v_0, v_2) + r(v_2, v_5) + r(v_5, v_8) \\ &\quad + r(v_8, v_{11} + r(v_{11}, v_0) \\ &\quad = 5 + 4 + 3 + 10 + 5 = 27. \end{aligned}$$

$$\begin{aligned} LR(R_2) &= r(v_0, v_1) + r(v_1, v_4) + r(v_4, v_7) \\ &\quad + r(v_7, v_{10}) + r(v_{10}, v_0) \\ &\quad 5 + 8 + 6 + 10 + 7 = 36. \end{aligned}$$

$$\begin{aligned} LR(R_3) &= r(v_0, v_3) + r(v_3, v_6) \\ &\quad + r(v_6, v_9) + r(v_9, v_{12}) + r(v_{12}, v_0) \\ &\quad = 8 + 5 + 5 + 6 + 7 = 31. \end{aligned}$$

$$LR(T) = 27 + 36 + 31 = 94.$$

The solution is feasible because the total resource consumption of all routes $LR(R_i)$ $(i = 1, ..., 3)$ is less than RM_{max}.

4.2 Literature Review

As we know, SBDP-RC has not been studied much. In the literature, several variants of the problem have been proposed; we describe these as follows: 1) mDMP or mTRP is the case when the resources are infinitive. Several metaheuristics for solving the problem were proposed in [4, 31, 28]. The experimental results showed that several algorithms [4, 31, 28] gave good solutions fast for large instances of up to 500 customers; 2) mTRP with distance constraint (mTRP-DC) is the case where the maximum duration of each vehicle is lower a predetermined value. The two metaheuristic algorithms in [7, 25] can be applied well to the problem in a reasonable amount of time; 3) Capacitated mTRP [22, 37] is the case where the vehicle's capacity does not exceed the permitted limit. The metaheuristic in [7] also receives good feasible solutions fast; 4) mTRP with profits (mTRPP) aims to find a solution to maximize the total revenue. In this case, some vertices may not be visited. The metaheuristic algorithm in [32, 24] produced good instances with up to 200 vertices; 5) The deliveryman problem (DMP) with (without) time windows is a special case of mTRP where there is a only vehicle to run. Numerous metaheuristic algorithms [5, 3, 6, 27] for the problem have also been developed. The experimental results showed their expressive performance for large instances; 6) Recently, a new variant of $mTRP$ post-disaster was introduced in [1, 9]. In this case, an additional cost for a road-clearance operator is involved in the function cost. They tested their algorithms on the Istanbul data set.

To our knowledge, the above algorithms are the best algorithms for the problem's several variants. However, resource constraints are not involved; therefore, these algorithms are not easily adapted to SBDP-RC.

4.3 Our Algorithm and Contribution

The problem can be solved by exact and heuristic (or metaheuristic) algorithms. An exact algorithm obtains an optimal solution, but it consumes much time. Heuristic approaches include the classical heuristic and metaheuristic algorithms: the former finds a solution fast, but the solution's quality may not be good; on the other hand, the latter reaches a near-optimal solution in a short amount of computation time. Therefore, metaheuristic is a suitable approach for solving large-scale problems; however, its efficiency is mainly evaluated through experiments.

A good metaheuristic needs to maintain a balance between exploration and exploitation strategies. The main contributions of this work can be summarized as follows:

- From an algorithmic perspective, the proposed metaheuristic consists of two phases: 1) in the first phase (the construction phase), an initial solution is created based on the insertion heuristic scheme. This step aims to obtain a good-enough solution; 2) the post-phase (the improvement phase) improves the

solution created from the previous one. Starting from a good-enough solution helps the algorithm to increase the chance of improving the solution's quality. In this phase, we use the randomized variable neighborhood search scheme (RVNS) to investigate various neighboring solutions to find good solutions. RVNS aims to exploit a good solution space that is explored. Two additional characteristics are integrated into the proposed algorithm. First, according to a distance metric, the algorithm accepts a solution that is worse than the current solution if it is far from it; this enhances the exploration of far-away valleys. Second, the search is allowed to move to unfeasible solution spaces by using a penalty method. When a constraint is violated, the value of the parameter increases to drive the search toward feasible regions. This means that the algorithm tries to exploit the feasible regions that are explored. After this, we enlarge the search space by decreasing it if no better solutions can be found. By doing this, the algorithm has a higher chance of finding better solutions. When the algorithm still fails in finding better solutions, the shaking technique is applied to move the search toward a completely new solution space that is unexplored.

- From the computational perspective, our algorithm obtains good feasible solutions fast for instances with large sizes. Additionally, the algorithm receives better solutions as compared to the previous algorithms in many cases.

The rest of this paper is organized as follows. Section 2 introduces our algorithm; then, the experiments are described in Section 3. Sections 4 and 5 discuss and conclude the article, respectively.

5 Proposed Algorithm

5.1 Variants of VNS

We describe VNS, GVNS [28], and shaking [27], respectively.

- VNS is described in [28]. It is divided into two main steps: 1) shaking, and a local search step. In the step, shaking implements the move to a random solution. The second phase consists of applying a local search to the solution and selecting the best one in a neighborhood set.
- Randomized VNS (RVNS) [28] is a variant of VNS. In RVNS, the search procedure is performed randomly to generate neighbor solutions.
- GVNS [28] is a variant of VNS. GVNS is a version of VNS in which VNS is applied as the improvement procedure. In this article, we use GVNS with a random neighborhood search.
- Skewed-GVNS [28] is an extension of basic GVNS that explores solution spaces that are far from the incumbent solution. Therefore, we can accept worse solutions if they are different from the incumbent.

5.2 Neighborhood Investigation

Several neighborhoods [13, 29] in the literature are applied to exploit the search region. Let $N_k(k = 1, ..., k_m)$ be a set of neighborhood structures. Now, let $T = (R_1, R_2, ..., R_l)$ be a tour with l routes, we then introduce a novel neighborhood structure.

For inter-route: it optimizes a route.

- **Forward** (N_1) pushes a vertex forward one position. This neighborhood of R is defined as a set $N_1(R) = \{R_i = (v_1, v_2, ..., v_{i-1}, v_{i+1}, v_i, ..., v_m) : i = 2, 3, ..., m-1\}$. The complexity time is $O(n)$.
- **Backward** (N_2) pushes a vertex backward one position. This neighborhood of R is defined as a set $N_2(R) = \{R_i = (v_1, v_2, ..., v_i, v_{i-1}, v_{i+1}, ..., v_m) : i = 2, 3, ..., m-1\}$. The complexity time is $O(n)$.
- **Exchange-adjacent** (N_3) exchanges each pair of adjacent vertices. This neighborhood of R is defined as a set $N_3(R) = \{R_i = (v_1, v_2, ..., v_{i-2}, v_i, v_{i-1}, v_{i+1}, ..., v_m) : i = 3, 4, ..., m-1\}$. The complexity time is $O(n)$.
- **Exchange** (N_4) exchanges the positions of each pair of vertices. This neighborhood of R is defined as a set $N_4(R) = \{R_{ij} = (v_1, v_2, ..., v_{i-1}, v_j, v_{i+1}, ..., v_{j-1}, v_i, v_{j+1}, ..., v_m) : i = 2, 3, ..., m-3; j = i+3, ..., m\}$. The complexity time is $O(n^2)$.
- **2-opt** (N_5) removes each pair of edges from the tour and reconnects them. This neighborhood of T is defined as a set $N_5(T) = \{T_{ij} = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+2}, v_{i+1}, v_{j+1}, ..., v_m) : i = 1, ..., n-4; j = i+4, ..., m\}$. The complexity time is $O(n^2)$.
- **3-opt** (N_6) reallocates three vertices to another position. This neighborhood of R is defined as a set $N_6(R) = \{R_i = (v_1, v_2, ..., v_{i-1}, v_i, v_{j+1}, ..., v_k, v_{i+1}, ..., v_j, v_{k+1}...., v_m) : i = 2, 3, ..., m-5, j = 4, ..., m-3, k = 6, ..., m-1\}$. The complexity time is $O(n^3)$.

For intra-route: Intra-route is used to swap or exchange vertices between two different routes.

- **Exchange-2-routes** $N_7(R)$ exchanges two vertices from different routes. The swap-2-route neighborhood of R_l and R_h is defined as a set $N_8(T) = \{T_i = (R_1, ..., R_2, ..., R_l = (v_{1l}, v_{2l}, ..., v_{ih}, v_{il+1}, ..., v_{ml}), ..., R_h = (v_{1h}, v_{2h}, ..., v_{il}, v_{ih+1}, ..., v_{mh}), ..., R_k) : il = 2, 3, ..., ml-1, ih = 2, 3, ..., mh-1\}$. The complexity time is $O(n^2)$
- **Insert-2-routes** $N_8(R)$ removes a vertex in turn and inserts it at the best possible position in the other. An insert-2-route neighborhood of R_l and R_h is defined as a set $N_8(T) = \{T_i = (R_1, ..., R_2, ..., R_l = (v_{1l}, v_{2l}, ..., v_{ih-1}, v_{ih}, v_{il+1}, ..., v_{ml}), ..., R_h = (v_{1h}, v_{2h}, ..., v_{ih-1}, v_{ih+1}, ..., v_{mh}), ..., R_k) : il = 2, 3, ..., ml-1, ih = 2, 3, ..., mh-1\}$. The complexity time is $O(n^2)$.

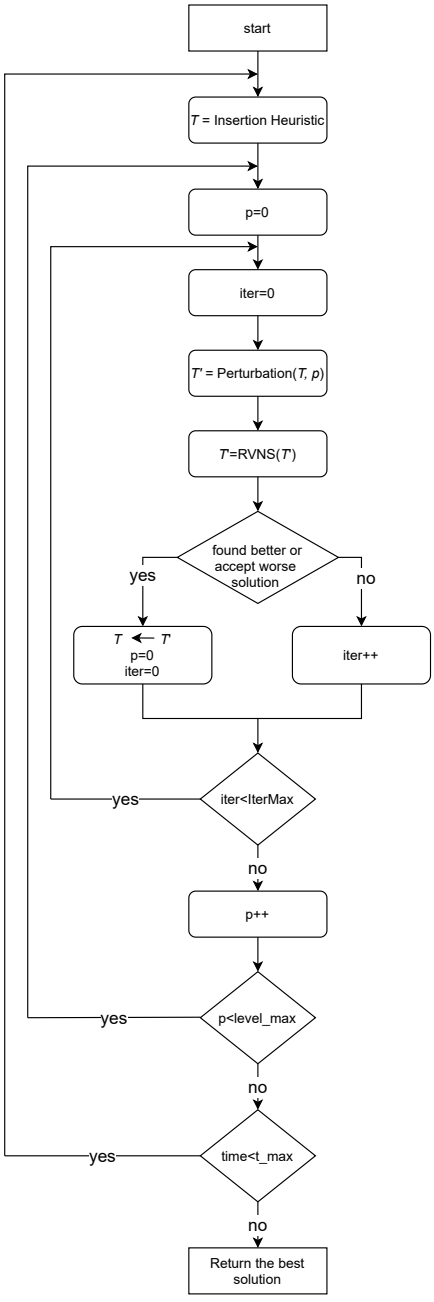
5.3 Restricted Infeasible Solution Space

Infeasible solutions are penalized by a value. With route T, let$VS(T), LR(R_i)$ be a penalty value and the resource consumption of route R_i. Penalty value $VS(T)$ is computed as follows: $\max\{(\sum_{i=1}^{k} LR(R_i) - RM_{max}), 0\}$. The solutions are then calculated in accordance with $W' = W + PV \times VS(T)$, in which PV is the penalty factor. If the solution is feasible, then $LR \leq RM_{max}$ and $W' = W$. To make the algorithm's structure more readable, a flowchart of the proposed algorithm is described in Figure 4. The proposed algorithm consists of two phases. Algorithm 1 depicts the whole process in pseudocode.

5.4 Construction

Algorithm 2 shows the constructive procedure. Assume that we have a partial solution and $V^{'}$ is a list of unvisited vertices ($V^{'} \subseteq V$). To complete the partial solution, a

Figure 4: Flowchart of skewed GVNS algorithm



Algorithm 1 Skewed GVNS

Require: $T, IterMax$, $lvel\_max$, and t_{max} are a starting solution, the number of iterations, the strength of the perturbation procedure, and the maximum time to run, respectively.

Ensure: the best-found solution T^*.

repeat
  {Step 1: construction step}
  $T \leftarrow$ **Construction**(v_1, V);{ T is an initial solution. It can be feasible or infeasible}
  $p = 1$;
  {Step 2: improvement step}
  while $(p < lvel\_max)$ **do**
    $iter = 0$;
    while $(iter < IterMax)$ **do**
      $T^{'} \leftarrow T$;
      {driving the search to a new promising solution space}
      $T^{'} \leftarrow$ Perturbation(T,p);
      {implement RVNS to exploit good solution space}
      $T^{'} =$ RVNS($T^{'}$);
      {accepting the worse solution}
      if $(W(T^{'}) < W(T) \times (1 + \beta \times d(T^{'}, T))$ or $(W(T^{'}) < W(T^*))$ **then**
        $T \leftarrow T^{'}$;
        p=0;
        $iter$=0;
        {update best solution}
        if $((W(T^{'}) < W(T^*))$ and (T is feasible)) **then**
          $T^* \leftarrow T^{'}$;
        end if
      else
        $iter++$;
      end if
    end while
    $p++$;
  end while
until $time < t_{max}$
return T^*; =0

vertex in $V^{'}$ needs to be inserted. We need to select a vertex and the position to insert it into the solution. We use a greedy scheme to pick a vertex so that its insertion causes the solution with the lowest cost. A solution is generated when all of the vertices of K_n are routed. The procedure then returns the feasible solution (if any). Otherwise, for added randomness in routing, it tries to generate n solutions; then, the one with the minimum fitness value will be returned.

Algorithm 2 Construction(v_1, K_n)

Require: v_1, K_n are a main depot and the graph, respectively.
Ensure: A starting solution T.

$S = \emptyset$; {S is the list of infeasible routes}
FOUND=False;
$T = \phi$; {Initially, T is empty}
for $(l = 1; l \leq k; l++)$ **do**
  $R_l \leftarrow R_l \cup v_1$; {$k$ routes start at depot}
end for
repeat
  repeat
    Select a random route $R_l (R_l \in R)$;
    Randomly pick a new vertex v and an inserted position $j < |R_l|$ so that the cost of $R_l^{'}$ after inserting is minimal; {$|R_l|$ is the number of vertices inR_l}
    Update R_l by $R_l^{'}$;
  until all vertices are visited
  for $(i = 1; i \leq k; i++)$ **do**
    $T \leftarrow T \cup R_i$;{update all routes in the tour}
  end for
  if (T is feasible) **then**
    return T;
  else
    $S \leftarrow S \cup T$;
  end if
until $|S| < n$
if *FOUND*=False **then**
  $T \longleftarrow$ solution with minimum cost W' in set S;
end if
return T; =0

5.5 Improvement

In the second step, it tries to improve the feasible solution that was created by the previous phase. In this step, we use RVNS in [28] to exploit the neighborhood solutions. Whenever a given neighborhood of set N fails to improve the current best solution, RVNS randomly selects another neighborhood from the same set to continue the search. The aim of using RVNS is to exploit a good solution space that has just been explored. Preliminary experiments indicate that randomly selecting another neighborhood can find better solutions than a deterministic order. If we find a better solution, it becomes the new current solution. However, the search cannot escape from very large valleys in some cases. In this paper, we adopt a skewed VNS approach that permits moves to worse solutions to explore more valleys that are far from the current solution. The aim is to support the search for getting out of huge valleys. Here, we

Algorithm 3 RVNS(T)

Require: T is a route.
Ensure: A new solution T.
1: Initialize neighborhood list NL;
2: **while** $NL \neq 0$ **do**
3: Choose a neighborhood N in NL at random
4: $T^{'} \leftarrow$ arg min $N(T)$; {Neighborhood search}
5: **if** $((L(T^{'}) < L(T))$ **then**
6: $T \leftarrow T^{'}$
7: Update NL;
8: **else**
9: Remove N from the NL;
10: **end if**
11: **end while**=0

Algorithm 4 Perturbation(T, p)

Require: T, T^{*}, p are the route, the best current route, and the value to control the strength of the perturbation, respectively.
Ensure: a route T.
1: $i = 1$;
2: **while** $(i < p)$ **do**
3: {Select random method to shaking}
4: rnd=rand(2);
5: **if** $(rnd == 1)$ **then**
6: $T^{'} \leftarrow$ Apply double-bridge in T;
7: **else**
8: $T^{'} \leftarrow$ Exchange randomly vertices in T;
9: **end if**
10: $T^{''} \leftarrow$ arg min 3-opt-($T^{'}$);
11: **if** $(W(T^{''}) > (1 - \rho) \times W(T^{*}))$ **then**
12: $T \leftarrow T^{''}$
13: break;
14: **else**
15: $i++$;
16: **end if**
17: **end while**
18: **return** T; =0

make a move from solution T to neighboring solution $T^{''}$ if

$$W(T^{'}) < W(T) \times (1 + \beta \times d(T, T^{'}). \tag{6}$$

Let $d(T, T^{'})$ be the metric distance between T, and $T^{'}$; this shows the difference between the two solutions. The greater and greater the metric distance is, the more and more the difference is. In mathematical respect, the distance is the minimum

number of transformations from T to $T^{'}$. When there exists no polynomial operator for calculating $d(T, T^{'})$, $d(T, T^{'})$ is n (the number of vertices in the graph) minus the number of vertices that have the same position in both T and $T^{'}$.
The detail of the improvement step is described in Algorithm 3.

The aim of the perturbation mechanism is to maintain exploration; it drives the search to a new promising solution space. If the mechanism implements too-small shaking moves, the search gets stuck into the local optima. Conversely, large moves in the shaking drive the search to unpromising or infeasible spaces. In an approach to fulfill the omission, we use a new shaking technique that was developed from the original double-bridge technique [27]. A randomly neighboring solution $T^{''}$ is generated by the double-bridge or random exchange method; then, it replaces the current solution if $(W(T^{''}) > (1-\rho) \times W(T^{*}))$ (ρ is a threshold ratio). The shaking procedure performs p times, where ρ is a parameter that is called the strength of the shake. The shaking is applied successfully in [37]. The detail is described in Algorithm 4.

5.6 Independent Component Analysis

Independent component analysis (ICA) is a statistical technique used to separate independent sources that affect data. In the context of EEG the set of independent components can be interpreted as the sources of brain activity that generated the recorded signal. ICA belongs to a family of methods called blind source separation (BSS) [8].

The BSS problem can be represented by the equation:

$$\mathbf{S} = \mathbf{W}\mathbf{X}, \tag{7}$$

where $\mathbf{S} \in \mathbb{R}^{CxM}$ is the matrix of C components for M samples, $\mathbf{W} \in \mathbb{R}^{CxN}$ is the transition matrix with the weight vectors between each signal and electrode and $\mathbf{X} \in \mathbb{R}^{NxM}$ is the data from N electrodes. W is the unknown separation matrix that will satisfy this equation.

The limitation of the BSS methods consists in impossible determination of the original amplitude of the source signals and no more than $\mathbb{N}$ sources can be found for $\mathbb{N}$ recorded signals [20]. Additionally, independent component analysis (ICA) relies on the assumption that the sources are statistically independent, indicating that the sources are not correlated with each other and as a rule not distributed in the value domain. If a distribution of the original signals is close to normal, the result can be ambiguous [20]. The goal with a matrix $\mathbb{S}$ is to find the components of $\mathbb{X}$ that are as independent of each other as possible. To obtain this, the data is pre-processed, specifically centered (the mean of each signal is zero) and whitened (variance of each signal is equal to 1) to remove correlation. Then, different measures of normality (negentropy or kurtosis) are used to modify a $\mathbb{W}$ matrix using the Newton approximation method and the chosen non-quadratic function.

There are many variations of the ICA algorithm. However, the fastICA algorithm was chosen for the implementation because it is the most commonly used and it was easy to use parallel methods on it.

5.7 Data Representation and Implementation

The fastICA algorithm implementation was written in C, based on the version available in Matlab and the open-source it++ library. The *tanh* (hyperbolic tangent) function

Listing 1: Scheme of fastICA algorithm

```
// W - separating matrix
// n - number of eletrodes, m - number of samples
// resultW - the resulting separation matrix
// whiteningMatrix - calculated earlier
  randWeight(W, n, n); // random weights
  orth(W, n, n); // orthogonalization
  while (!found) { // next iterations
    found=0;
    if (it > maxNumIterations - 1) {
      //iteration count exceeded
    }
    it++;
    // Weight matrix normalization and convergence estimation
    // Checking if the weights have changed
    if(!found) {
      // Modification of weights and set found parameter
    }
  }
  // Rewriting the final form of the separating matrix
  mul(W, n, true, whiteningMatrix, n, false, n, resultW);
```

from the basic mathematical library was used to modify the weights. A symmetrical approach of searching for components was adopted, indicating that the algorithm calculates all components simultaneously.

In Listing 1 there is a scheme of fastICA algorithm with an indication of the most important steps. The *mul* functions use matrix multiplication on the CPU or GPU depending on the version of the implementation.

The *parallel* directives from OpenMP were used in parts of the algorithm where the entire signal was applied (signal whitening and calculation of successive approximations). With the Intel Cilk Plus extensions for C and C++, one can use array notation and built-in reduction functions (such as finding the maximum or minimum value in an array), which makes the code more readable and forces effective vectorization. For parts of the algorithm, such as matrix multiplication, vector and eigenvalue calculations, it was much more advantageous to use ready-made solutions from the BLAS and MKL libraries (*cblas_dgemm*, *cblas_dcopy*, *LAPACKE_dsyev*, *LAPACKE_dgesvd*) than their implementations [33].

Originally, in the iterative part of the algorithm, the matrix multiplication of the entire signal was implemented by multiplication functions from the CUBLAS library, which is the equivalent of the BLAS library for the NVIDIA graphics cards. When performing calculations on the graphics card, it is important to be aware of the additional time overhead associated with host-device data transfer [41]. In this situation, each iteration means additional operations related to uploading and downloading data. Using the potential of the GPU inside the iterative loop, each data transfer seemed risky. However, by transferring the calculations related to the use of the non-square

function and modifying the weights to a separate library compiled by nvcc, the size of the data sent at a time was decreased. This required writing a function dedicated to the graphics card (kernel) taking into account the problems of memory shared by blocks and threads of the CUDA architecture. In Listing 2, you can find a code fragment along with its invocation. On the other hand, in Listing 3, there is a section of C code that executes in each iteration.

6 Results

All tests performed on these architectures are described in Table 4.

Table 4: Selected architectures

Infrastructure	**Cluster name**	**CPU**	**Number of cores**	**GPU**
UMCS	Solaris	2x Intel Xeon E5-2670 v3 @ 2.30GHz	24	NVIDIA Tesla V100s
PLGrid	Zeus	2x Intel Xeon X5650 @ 2.67GHz	12	NVIDIA Tesla M2090
PLGrid	Prometheus	2x Intel Xeon E5-2680 @ 2.5GHz	24	NVIDIA Tesla K40d

The study compared the speed of the fastICA implementation for the data sets of 256×1000 (1 second of recording), 256×10000 (10 seconds of recording), 256×100000 (100 seconds of recording) and 256×1000000 (1000 seconds of recording) data sets. For each architecture, the time efficiency was compared depending on the number of threads the program was launched with. The initial selection of weights was the same each time, and the resulting time was averaged over the number of needed iterations (for the purposes of this study, it was established that approximately 100 iterations were needed for the selected EEG data).

6.1 Tests of implementation

The previous research proved that the use of virtual cores does not speed up the performance of the algorithm as is the case with physical cores, so the tests were performed without the use of hyper-threading. [14, 15, 17, 18].

6.1.1 Solaris

Table 5 shows the program execution time in seconds for all data. Figure 5 presents the acceleration obtained by the CPU+GPU version compared to the CPU version with a given number of threads. As follows from the results the use of GPU capabilities speeds up the performance of the algorithm. The difference is not as significant as the

Listing 2: Code for kernel function

```
__global__ void kernelSum(double * __restrict__ A,
    double * __restrict__ blockResults, int m, double a1) {

  extern __shared__ double sums[];
  double sum = 0.0;
  double * p = &A[blockIdx.x * m];

  for(int i = threadIdx.x; i < m; i += blockDim.x) {
    // Using of tangent functions
    A[blockIdx.x * m + i] = tanh(a1*p[i]);
    sum += (1 - pow(p[i], 2));
  }

  sums[threadIdx.x] = x;
  __syncthreads();

  for(int offset = blockDim.x / 2; offset > 0; offset >>= 1) {
    if(threadIdx.x < offset) {
      sums[threadIdx.x] += sums[threadIdx.x + offset];
    }
    __syncthreads();
  }

  if(threadIdx.x == 0) {
    blockResults[blockIdx.x] = sums[0];
  }
}

void tanhGPU_wrapper(double * A, double a1, int n, int m,
    double * X, double * W, double * G, double * nvec) {

  int nm = n*m;
  int bs, mings;

  cudaOccupancyMaxPotentialBlockSize(
        &mings, &bs, tanhGPU, 0, nm);
  size_t shmsize = sizeof(double) * bs;

  mulGPU(X, m, true, W, n, false, n, A);

  kernelSum<double><<<n, bs, shmsize>>>(A, nvec, m, a1);

  mulGPU(X, n, false, A, n, false, m, G);

  cudaDeviceSynchronize();
}
```

Listing 3: Part of the code with the modification of weights using the kernel function

```
1  // W - buffer with weights
2  // c_W - buffer with weights for CUDA
3  cublasSetVector(nn, sizeof(*W), W, 1, c_W, 1);
4
5  // n - size; W is matrix nxn
6  copyMatrix(Wprev, W, n, n);
7
8  cublasSetVector(n, sizeof(*nvec), nvec, 1, c_nvec, 1);
9
10 // Formula for maximizing the normal distribution
11 // using the tangent function
12 // m - number of samples
13 // G1, G2 -- values of tangent functions
14 // c_X - input values
15 // c_hypTan - matrix with results from kernel function
16 // nvec - vector with results from kernel funtion
17
18 tanhGPU_wrapper(c_hypTan, a1, n, m, c_X, c_W, c_G1, c_nvec);
19 cublasGetVector(nn, sizeof(*c_G1), c_G1, 1, G1, 1);
20 cublasGetVector(n, sizeof(*c_nvec), c_nvec, 1, nvec,1);
21
22 #pragma omp parallel // efficiency: 95\%,
23                      // gain: 1,90x according to Intel Advisor
24 {
25 #pragma omp for
26   for(int i = 0; i < n; i++)
27     G2[i*n:n] = W[i*n:n]*nvec[i];
28 } //parallel
29
30 // Modification of weights
31 W[0:nn] = (G1[0:nn] - G2[0:nn])/m*a1;
```

Table 5: Solaris: 2x Intel Xeon CPU E5-2670 v3 @ 2.30GHz – 24 cores, NVIDIA Tesla V100s

	256x1000		**256x10000**		**256x100000**		**256x1000000**	
Number of threads	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**
1	3.075	2.781	4.118	11.577	5.100	100.253	17.346	989.861
2	3.197	2.582	4.150	8.154	5.253	64.152	15.541	648.398
3	3.346	2.456	4.479	7.572	4.860	57.090	14.205	571.254
4	2.862	2.087	3.835	6.379	4.337	48.028	13.069	461.153
5	2.827	2.338	3.942	6.511	5.107	49.054	13.467	459.481
6	3.026	1.992	3.813	5.738	4.571	41.999	13.278	409.896
7	2.934	1.958	3.944	5.919	4.358	45.687	12.802	404.973
8	2.721	1.918	3.716	5.291	4.530	39.361	12.306	382.040
9	2.982	2.039	4.237	5.643	4.486	42.711	12.386	390.353
10	2.828	1.977	4.213	5.440	4.294	38.645	12.465	364.328
11	2.879	2.003	3.946	5.603	4.666	40.752	12.361	375.422
12	2.880	1.925	3.783	5.448	4.429	40.369	12.562	354.141
13	2.799	1.976	3.887	5.379	4.352	42.271	12.426	368.260
14	2.861	1.981	4.246	5.256	4.468	38.177	12.124	346.272
15	2.886	1.925	3.804	5.474	4.429	40.127	12.181	350.959
16	2.705	1.821	4.008	5.215	4.099	36.894	12.385	341.640
17	2.948	1.997	3.886	5.658	4.415	38.647	12.524	348.708
18	2.890	2.013	3.938	5.347	4.392	37.018	12.099	340.452
19	2.903	2.029	3.946	5.496	4.423	40.451	12.392	341.226
20	3.067	2.066	3.802	5.360	4.443	37.483	12.184	351.473
21	2.900	1.949	3.958	5.435	4.403	37.311	12.310	339.592
22	2.837	2.076	4.060	5.471	4.357	38.132	12.012	342.490
23	2.998	2.124	3.904	5.475	4.422	39.912	12.412	341.577
24	3.085	2.106	4.052	5.439	4.450	37.579	12.189	346.825

number of threads increases, but there is still a slight time gain from using the extra CPU cores.

Figure 6 shows the acceleration of the CPU version (a) and CPU+GPU (b) depending on the number of threads. The speedup is calculated in relation to a single-threaded program. They show more clearly that adding computational cores still generates a profit, although in the case of the CPU version the graphs are flatter.

Notably, the machine with the Tesla V100S card takes 17 seconds for the largest data size using a single thread. However, we are to do with very powerful equipment of the latest generation. It is worth noting, however, that the use of CUDA capabilities without introducing parallel calculations on the CPU reduces the calculation time significantly. It is important in the context of building computing units in the research centers for the processing of EEG data. Providing such a unit with a graphics card with adequate power can be cheaper than building a multi-core computing cluster.

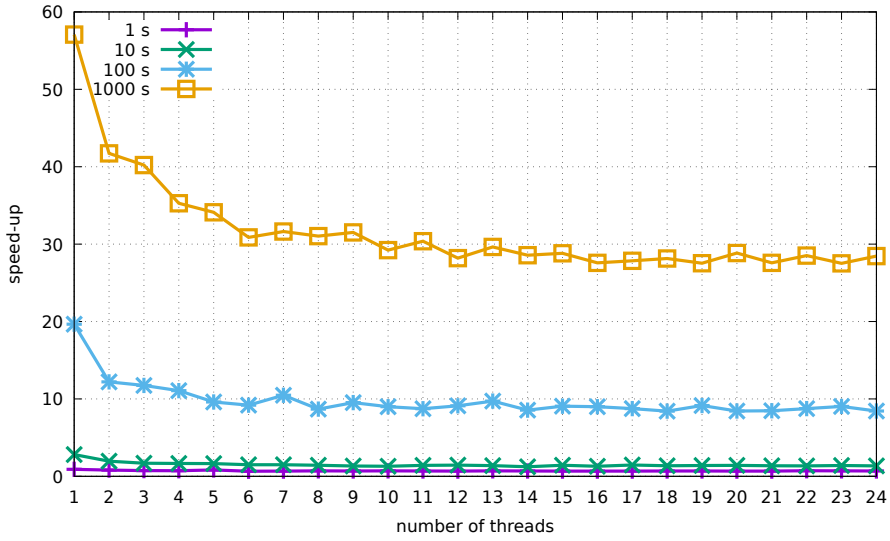


Figure 5: Solaris: 2x Xeon CPU E5-2670 v3 @ 2.30GHz – 24 cores, NVIDIA Tesla V100s – comparing the acceleration of the CPU+GPU version to the CPU

6.1.2 Zeus and Prometheus

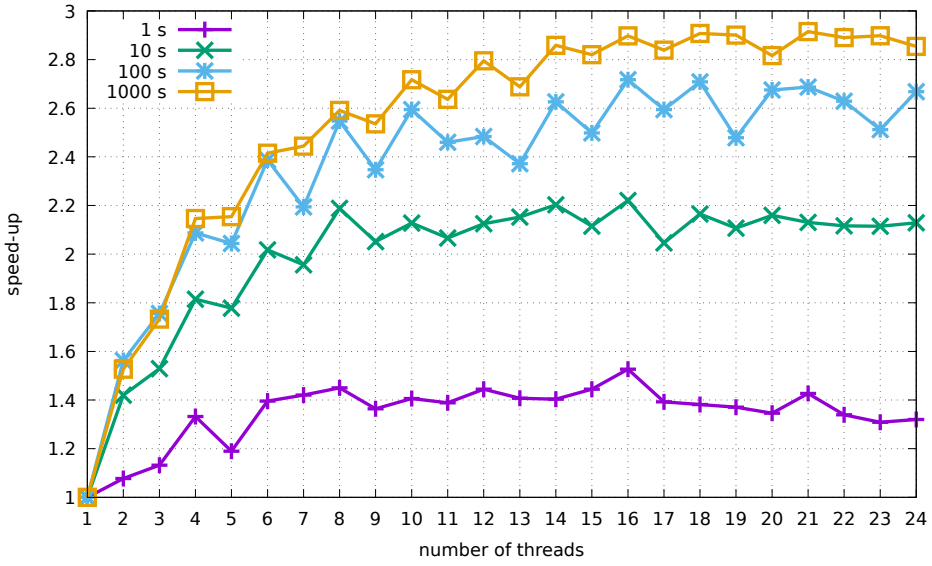
Tables 6 and 7 show the program execution time in seconds for all data. As in the Solaris cluster, more cores reduce the computation time, the implementation is similarly scalable, and the increase in data size generates better time gains.

Figures 7 and 9 present the acceleration of the CPU+GPU version compared to that of CPU with a given number of threads. Figures 8 and 10 show the obtained acceleration of the version of the implementation using CPU (a) and CPU+GPU (b) depending on the number of threads. The speedup is calculated in relation to a single-threaded program. One can see a similar relationship, that is, using more threads, you can still expect a gain in time.

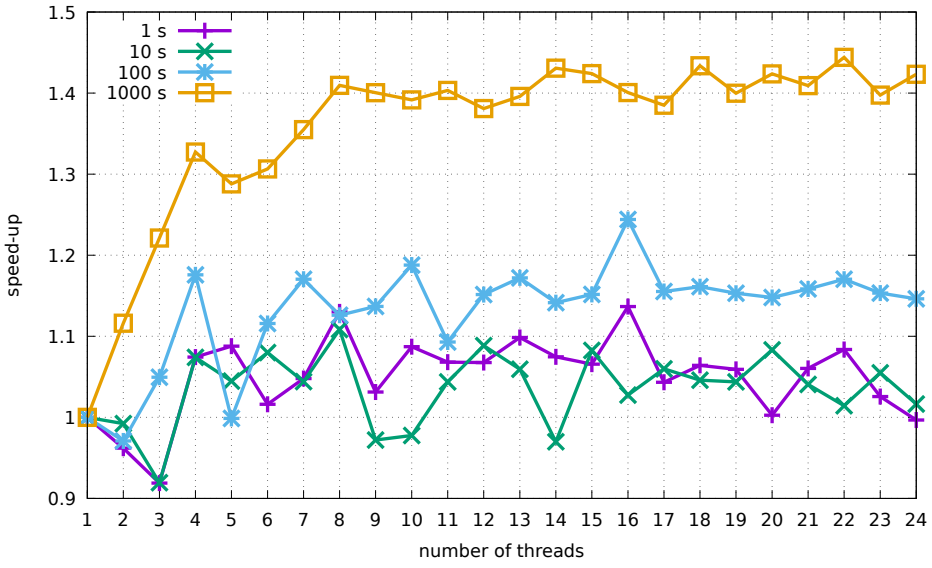
7 Discussion

The results proved that the support from the graphics card and CUDA technology accelerates the execution time of the algorithm significantly. The increase in the number of CPU cores still resulted in acceleration, although the benefits were not so visible.

Owing to the variety of architectures, it is possible to indicate which elements of the implementation are crucial for time optimization (in terms of reducing the running time) of the algorithm. At the same time, it is worth mentioning that due to the complexity of the structure of computing clusters, creating efficient programs on them is a demanding task. The fact that fastICA is an iterative method requires thread synchronization after each iteration, which affects the execution time significantly and makes it difficult to scale. It can be stated that this is the main ”bottleneck” of the algorithm with no prospects for improving the results at this point.



(a) CPU-based implementation



(b) CPU+GPU-based implementation

Figure 6: Solaris: 2x Intel Xeon CPU E5-2670 v3 @ 2.30GHz – 24 cores, NVIDIA Tesla V100s – the acceleration compared to the single-threaded version

As a matter of fact, the machine with the NVIDIA Tesla V100s graphics card gave the most satisfactory results, although cards of this type are not the cheapest product.

Table 6: Zeus: 2x Intel Xeon X5650 @ 2.67GHz – 12 cores, Tesla M2090

	256x1000		256x10000		256x100000		256x1000000	
Number of threads	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**
1	8.596	8.824	6.498	21.854	13.592	224.053	109.585	2242.188
2	6.994	6.436	5.408	13.451	12.222	130.692	101.576	1298.195
3	6.134	5.070	4.937	10.122	11.638	98.258	98.350	970.889
4	5.476	3.963	4.632	7.903	10.985	77.353	96.650	765.494
5	5.623	3.923	4.674	7.578	10.957	71.673	96.109	704.106
6	5.513	3.748	4.585	6.679	10.880	63.229	95.326	626.615
7	5.522	3.704	4.502	6.583	10.791	61.962	95.154	610.624
8	5.190	3.304	4.454	6.055	10.723	56.966	94.769	555.167
9	5.436	3.603	4.459	6.303	10.664	57.433	94.745	555.466
10	5.411	3.564	4.419	6.299	10.710	57.180	94.519	543.231
11	5.366	3.716	4.455	6.135	10.802	56.176	94.573	545.689
12	5.573	3.506	4.551	6.102	10.571	57.342	94.395	552.920

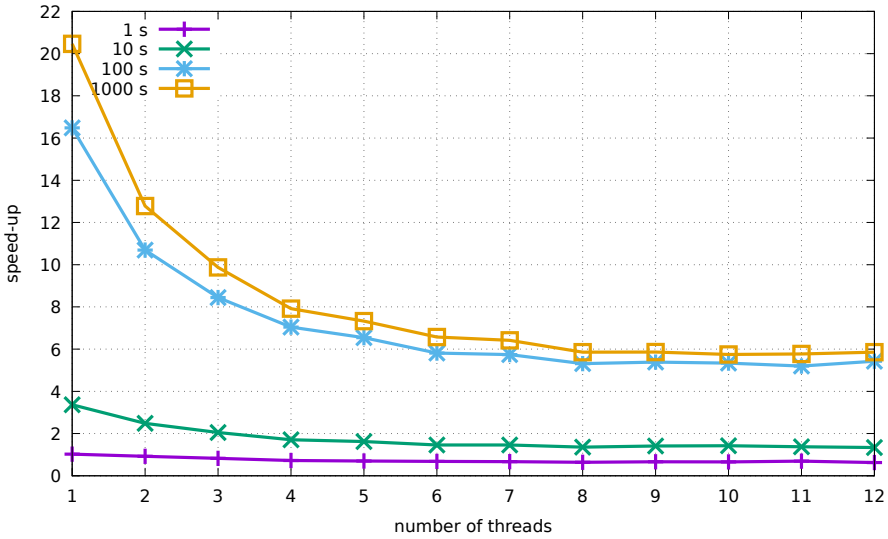
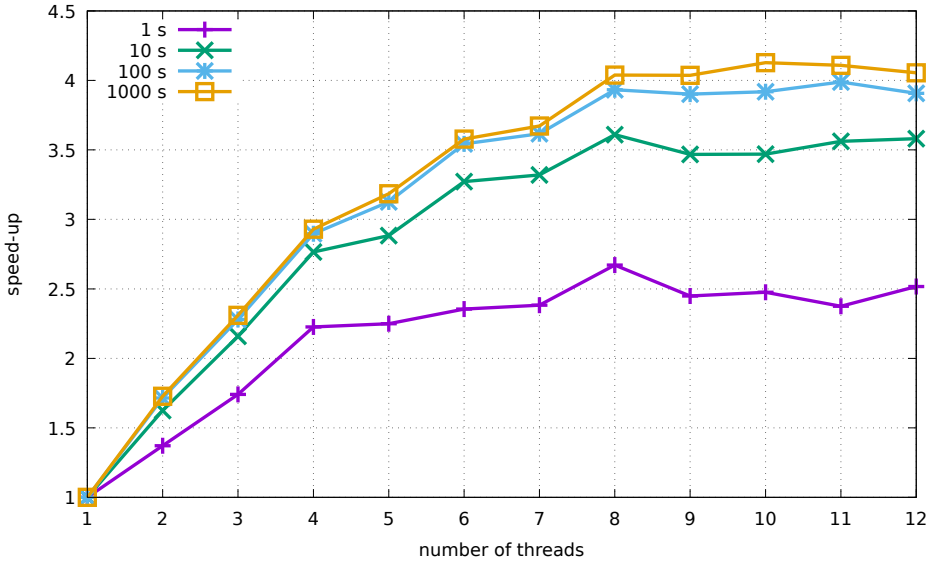
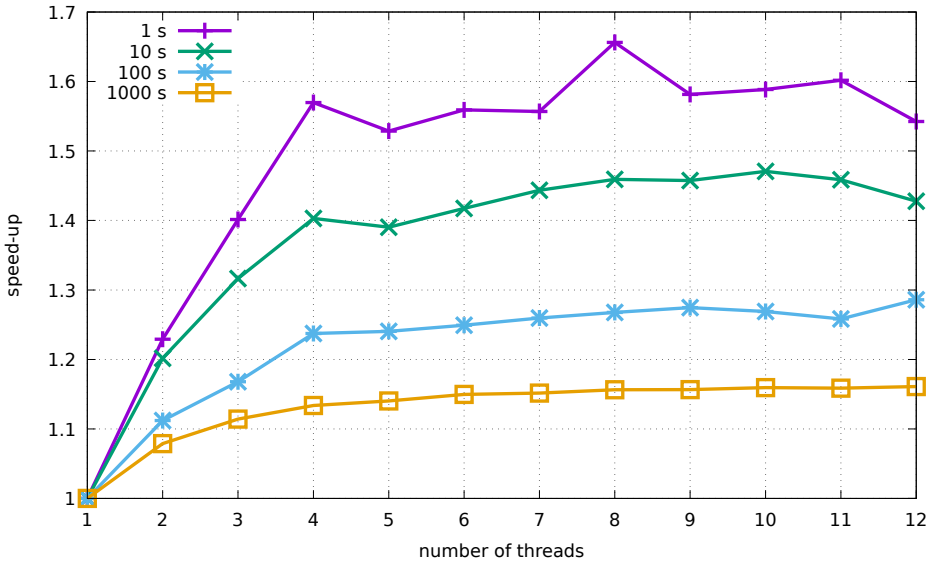


Figure 7: Zeus: 2x Intel Xeon X5650 @ 2.67GHz – 12 cores, Tesla M2090 – comparing the acceleration of the CPU+GPU version to the CPU

However, the latest generation Intel Xeon processors are also quite expensive. These factors should also be considered when building a computational machine for EEG data analysis. Providing such unit with a graphics card with the appropriate power could be less expensive than constructing a multi-core computing cluster.



(a) CPU-based implementation



(b) CPU+GPU-based implementation

Figure 8: Zeus: 2x Intel Xeon X5650 @ 2.67GHz – 12 cores, Tesla M2090 – the acceleration compared to the single-threaded version

8 Conclusions and future works

This paper presents the time execution results for a parallel version of the fastICA algorithm adapted to EEG signals and multi-core architectures (capacity of Intel ar-

Table 7: Prometheus: 2x Intel Xeon E5-2680 @ 2.5GHz – 24 cores, Tesla K40d

	256x1000		256x10000		256x100000		256x1000000	
Number of threads	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**	**CPU+ GPU**	**CPU**
1	2.494	2.575	1.470	6.272	4.328	63.333	43.934	633.295
2	2.144	2.340	1.448	4.237	4.184	40.630	42.545	399.542
3	2.057	2.223	1.459	3.724	4.154	33.801	42.229	353.096
4	1.968	2.056	1.405	3.242	4.059	29.936	41.688	300.085
5	2.089	2.150	1.421	3.299	4.099	28.971	41.608	281.807
6	1.951	2.083	1.366	3.091	4.029	26.651	41.509	259.350
7	1.911	2.042	1.352	3.073	4.064	28.096	41.478	271.448
8	1.733	1.937	1.305	2.798	4.026	24.679	41.283	237.445
9	1.934	1.956	1.343	2.985	3.967	25.829	41.204	246.441
10	1.886	1.984	1.361	2.859	4.042	24.261	41.231	232.248
11	1.866	1.899	1.338	2.819	3.970	24.460	41.245	232.758
12	1.869	1.917	1.409	2.737	3.965	24.172	41.082	216.436
13	1.885	1.941	1.285	2.836	3.935	23.769	41.180	234.071
14	1.879	1.864	1.319	2.783	4.006	23.355	41.048	223.344
15	1.907	2.054	1.387	2.857	3.968	24.689	41.112	215.137
16	1.912	1.861	1.318	2.803	4.002	22.574	41.096	207.854
17	1.871	1.874	1.371	2.937	4.019	24.664	40.990	241.892
18	1.892	1.993	1.377	2.854	3.991	22.576	41.045	219.282
19	1.940	2.132	1.330	2.920	3.957	25.215	41.052	227.162
20	1.975	2.142	1.449	2.828	4.159	23.150	40.995	205.798
21	2.103	2.042	1.335	2.992	3.936	23.164	41.055	216.180
22	1.959	1.954	1.316	2.924	3.921	22.844	40.982	204.775
23	1.975	2.167	1.385	3.063	3.982	24.487	41.162	226.745
24	2.243	2.080	1.483	2.893	3.992	22.750	41.086	213.694

chitectures and available libraries as well as extensions together with CUDA libraries). Tests were carried out on several computational clusters using the EEG data of various sizes. The paper indicates the possibility of improving the parallelized version of the algorithm by transferring more calculations to the graphics card.

The future plan is also to integrate existing solutions with EGI System NetStation and after separation of sources, to make an attempt to reject artifacts found in this way automatically. A possible method of doing this is to use, among others, convolutional neural networks.

Neuroimaging techniques, including EEG, provide excellent opportunities to get to know how a person functions, and thus, based on the research and diagnostics, to improve living conditions. Shortening the time of the electroencephalographic signal analysis tools, it is possible to help more patients, accelerate research, and improve BCI solutions.

Acknowledgements

This research was partially supported by ....

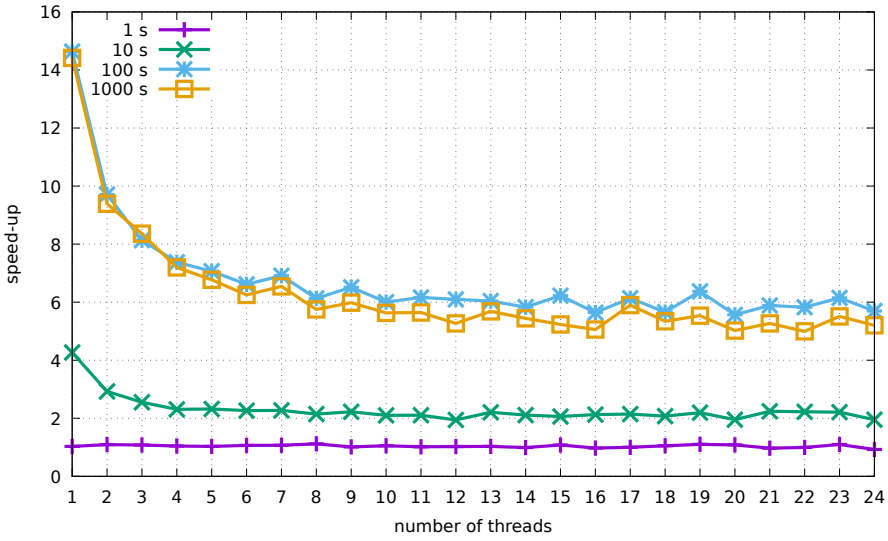


Figure 9: Prometheus: 2x Intel Xeon E5-2680 @ 2.5GHz – 24 cores, Tesla K40d – comparing the acceleration of the CPU+GPU version to the CPU

References

[1] M. Ajam, V. Akbari, and F. S. Salman. Minimizing latency in post-disaster road clearance operations. *European Journal of Operational Research*, 277(3):1098–1112, 2019.

[2] A. Albajes-Eizagirre, L. Dubreuil Vall, I.-S. David, A. Riera, A. Soria-Frisch, S. Dunne, and G. Ruffini. *EEG/ERP analysis: methods and applications*. CRC Press, 10 2014.

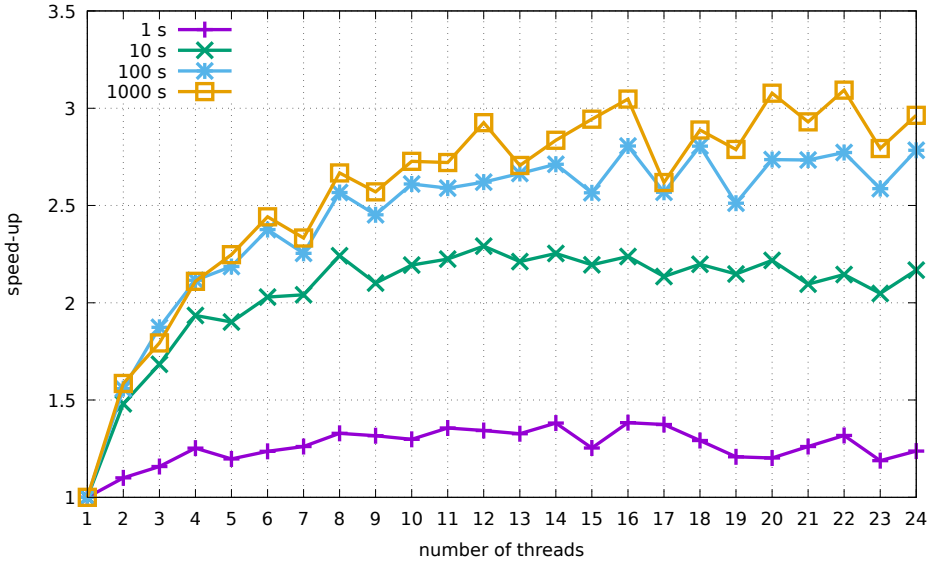
[3] H.-B. Ban. A rvnd+ils metaheuristic to solve the delivery man problem with time windows. In A. Tagarelli and H. Tong, editors, *Computational Data and Social Networks*, pages 63–69, Cham, 2019. Springer International Publishing.

[4] H. B. Ban and D. N. Nguyen. An effective grasp+vnd metaheuristic for the k-minimum latency problem. In *2016 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pages 31–36, 2016.

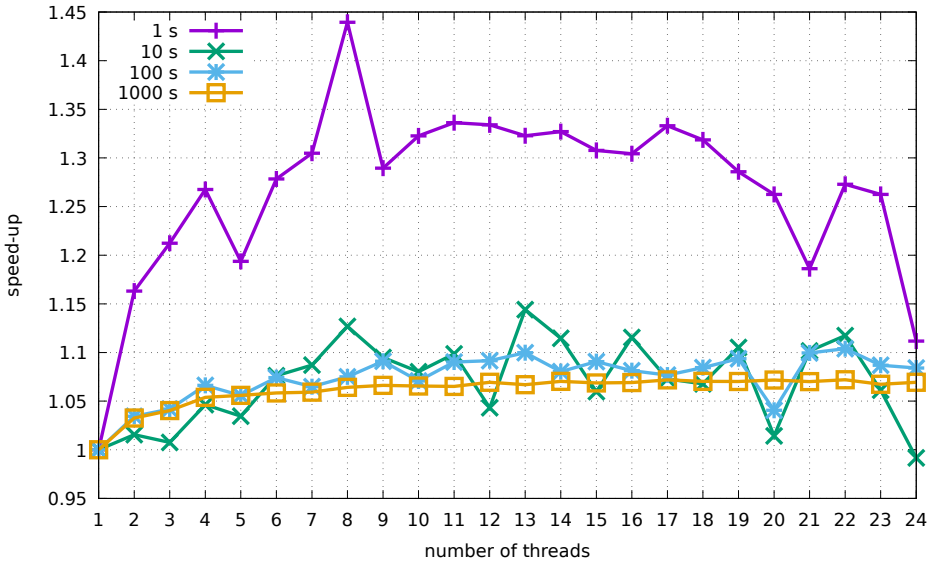
[5] H. B. Ban and D. N. Nguyen. A meta-heuristic algorithm combining between tabu and variable neighborhood search for the minimum latency problem. *Fundamenta Informaticae*, 156(1):21–41, 2017.

[6] H. B. Ban and D. N. Nguyen. Metaheuristic for the traveling repairman problem with time window constraints. In *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 1–6, 2019.

[7] B. H. Bang. A grasp+vnd algorithm for the multiple traveling repairman problem with distance constraints. *Journal of Computer Science and Cybernetics*, 33(3):272–288, Mar. 2018.



(a) CPU-based implementation



(b) CPU+GPU-based implementation

Figure 10: Prometheus: 2x Intel Xeon E5-2680 @ 2.5GHz – 24 cores, Tesla K40d – the acceleration compared to the single-threaded version

[8] G. D. Brown, S. Yamada, and T. J. Sejnowski. Independent component analysis at the neural cocktail party. *Trends in neurosciences*, 24(1):54–63, 2001.

[9] M. Bruni, P. Beraldi, and S. Khodaparasti. A hybrid reactive grasp heuristic for the risk-averse k-traveling repairman problem with profits. *Computers & Operations Research*, 115:104854, 2020.

[10] M.-C. de Marneffe, A. N. Rafferty, and C. D. Manning. Finding contradictions in text. In *Proceedings of ACL-08: HLT*, pages 1039–1047, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[11] A. Delorme, T. Sejnowski, and S. Makeig. Enhanced detection of artifacts in eeg data using higher-order statistics and independent component analysis. *Neuroimage*, 34(4):1443–1449, 2007.

[12] C. L. Dickter and P. D. Kieffaber. *EEG methods for the psychological sciences*. SAGE Knowledge, Los Angeles, 2014.

[13] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6:109–133, 1995.

[14] A. Gajos and G. M. Wójcik. Independent component analysis of eeg data for egi system. *Bio-Algorithms and Med-Systems*, 12(2):67–72, 2016.

[15] A. Gajos-Balińska, G. M. Wójcik, and P. Stpiczyński. Concept of independent component analysis algorithm parallelisation. In *Proceedings of Cracow Grid Workshop*, CGW'15, pages 55–56, 2015.

[16] A. Gajos-Balińska, G. M. Wójcik, and P. Stpiczyński. Parallel independent component analysis algorithm - performance comparison for eeg signal. In *Proceedings of Cracow Grid Workshop*, CGW'17, 2017.

[17] A. Gajos-Balińska, G. M. Wójcik, and P. Stpiczyński. High performance optimization of independent component analysis algorithm for eeg data. *Lecture Notes in Computer Science*, 10777:495–504, 2018.

[18] A. Gajos-Balińska, G. M. Wójcik, and P. Stpiczyński. Cooperation of cuda and intel multi-core architecture in the independent component analysis algorithm for eeg data. *Bio-Algorithms and Med-Systems*, 16(3), 2020.

[19] A. Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[20] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4):411–430, 2000.

[21] A. Kawala-Janik, W. Bauer, A. Al-Bakri, C. Haddix, R. Yuvaraj, K. Cichon, and W. Podraza. Implementation of low-pass fractional filtering for the purpose of analysis of electroencephalographic signals. In *Conference on Non-integer Order Calculus and Its Applications*, pages 63–73. Springer, 2017.

[22] L. Ke and Z. Feng. A two-phase metaheuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 40(2):633–638, 2013.

[23] A. Lastovetsky, L. Szustak, and R. Wyrzykowski. Model-based optimization of eulag kernel on intel xeon phi through load imbalancing. *IEEE Transactions on Parallel and Distributed Systems*, 28(3):787–797, 2016.

[24] Y. Lu, U. Benlic, Q. Wu, and B. Peng. Memetic algorithm for the multiple traveling repairman problem with profits. *Engineering Applications of Artificial Intelligence*, 80:35–47, 2019.

[25] Z. Luo, H. Qin, and A. Lim. Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints. *European Journal of Operational Research*, 234(1):49–60, 2014.

[26] M. Marelli, L. Bentivogli, M. Baroni, R. Bernardi, S. Menini, and R. Zamparelli. SemEval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 1–8, Dublin, Ireland, Aug. 2014. Association for Computational Linguistics.

[27] O. Martin, S. W. Otto, and E. W. Felten. *Large-step Markov chains for the traveling salesman problem*. Citeseer, 1991.

[28] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[29] N. Mladenović, D. Urošević, and S. Hanafi. Variable neighborhood search for the travelling deliveryman problem. *4OR*, 11:57–73, 2013.

[30] Netstation acquisition technical manual. documentation, egi, 2011.

[31] I. Ome Ezzine and S. Elloumi. Polynomial formulation and heuristic based approach for the k-travelling repairman problem. *International Journal of Mathematics in Operational Research*, 4(5):503–514, 2012.

[32] J. Pei, N. Mladenović, D. Urošević, J. Brimberg, and X. Liu. Solving the traveling repairman problem with profits: A novel variable neighborhood search approach. *Information Sciences*, 507:108–123, 2020.

[33] R. Rahman. *Intel Xeon Phi coprocessor architecture and tools: the guide for application developers*. Apress, Berkely, CA, USA, 2013.

[34] L. Szustak. Strategy for data-flow synchronizations in stencil parallel computations on multi-/manycore systems. *The Journal of Supercomputing*, 74(4):1534–1546, 2018.

[35] L. Szustak and P. Bratek. Performance portable parallel programming of heterogeneous stencils across shared-memory platforms with modern intel processors. *The International Journal of High Performance Computing Applications*, 33(3):534–553, 2019.

[36] M. Ungureanu, C. Bigan, R. Strungaru, and V. Lazarescu. Independent component analysis applied in biomedical signal processing. *Measurement Science Review*, 4(2):18, 2004.

[37] T. G. Will. *Extremal results and algorithms for degree sequences of graphs*. University of Illinois at Urbana-Champaign, 1993.

[38] G. M. Wojcik, J. Masiak, A. Kawiak, L. Kwasniewicz, P. Schneider, N. Polak, and A. Gajos-Balinska. Mapping the human brain in frequency band analysis of brain cortex electroencephalographic activity for selected psychiatric disorders. *Frontiers in Neuroinformatics*, 12, 2018.

[39] G. M. Wojcik, J. Masiak, A. Kawiak, L. Kwasniewicz, P. Schneider, F. Postepski, and A. Gajos-Balinska. Analysis of decision-making process using methods of quantitative electroencephalography and machine learning tools. *Frontiers in Neuroinformatics*, 13, 2019.

[40] G. M. Wojcik, J. Masiak, A. Kawiak, P. Schneider, L. Kwasniewicz, N. Polak, and A. Gajos-Balinska. New protocol for quantitative analysis of brain cortex electroencephalographic activity in patients with psychiatric disorders. *Frontiers in Neuroinformatics*, 12, 2018.

[41] R. Wyrzykowski, L. Szustak, and K. Rojek. Parallelization of 2d mpdata eulag algorithm on hybrid architectures with gpu accelerators. *Parallel Computing*, 40(8):425–447, 2014.