Computer Science • 26(4) 2025 https://doi.org/10.7494/csci.2025.26.4.6657

Dominik Żurek
Kamil Piętak
Marcin Pietroń
Marek Kisiel-Dorohinicki

A PARALLEL APPROACH FOR METAHEURISTICS SOLVING THE LABS PROBLEM USING CPU AND GPU

Abstract *This paper contributes to solving the low autocorrelation binary sequence (LABS) problem that remains an open hard-optimization problem with many applications. The current direction of research is focused on developing algorithms dedicated to parallel architectures such as GPGPU or multi-core CPUs. The paper follows this direction and proposes new heuristics developed from the steepest-descent local search algorithm that extends the notion of a neighborhood of a given sequence. The introduced algorithms utilize the parallel nature of multicore CPUs and provide an effective method for solving the LABS problem. The efficiency levels of SDSL and the new algorithm are presented; to ensure an effective comparison, they were both implemented in the same manner. The comparison shows that exploring the larger neighborhood improves the efficiency of the search method.*

Keywords LABS, parallel computing, steepest-descent local search, local optimization techniques

Citation Computer Science 26(4) 2025: 33–49

1. Introduction

This paper proposes new possibilities in metaheuristics construction for optimization problems that utilize parallel computing with CPU and GPU, concentrating on the low autocorrelation binary sequence as an example of a hard discrete problem.

Despite the extensive research, LABS remains an open optimization problem for long sequences. It has a wide range of applications, including communication engineering [18, 19, 21, 22], statistical mechanics [2, 14], and mathematics [10, 12]. For details of the various applications and the history of the problem, we refer to the existing source [16].

The low autocorrelation binary sequence is an NP-hard combinatorial problem with a simple formulation. It has been under intensive study since the 1960s by physics and artificial-intelligence communities. The LABS problem found its place in the CSPLIB list – a library of test problems for constraint solvers [20]. It consists of finding a binary sequence $S = \{s_0, s_1, \ldots, s_{L-1}\}$ with length L, where $s_i \in \{-1, 1\}$ minimizes energy function $E(S)$:

$$\begin{aligned} C_k(S) &= \sum_{i=0}^{L-k-1} s_i s_{i+k} \\ E(S) &= \sum_{k=1}^{L-1} C_k^2(S) \end{aligned} \tag{1}$$

Golay defined a so-called *merit factor* [9] that binds LABS energy level to the length of a given sequence:

$$F(S) = \frac{L^2}{2E(S)} \tag{2}$$

The search space for a problem of length L has size 2^L, and the energy of the sequence can be computed in time $O(L^2)$.

There are many different methods that have tried to solve the LABS problem. The simplest is an exhaustive enumeration (i.e., *the brute-force* method) that provides the best results, but can be applied only to small values of L. Some researchers use partial enumeration, choosing so-called skew-symmetric sequences [16] that are the most likely solutions for many lengths (e.g., for $L \in [31, 65]$, 21 best sequences are skew-symmetric). Additionally, the idea of branch-and-bound is applied, which assumes that a discrete optimization problem is solved by breaking up its feasible set into successively smaller subsets (branch), calculating bounds on the objective function value over each subset and using them to discard certain subsets from further consideration (bound) [15]. The best solution that can be found during this procedure is a global optimum. The goal is, of course, to discard many subsets as early as possible during the branching process; i.e., to discard most of the feasible solutions before actually evaluating them [16]. Enumerative algorithms (complete or partial) are limited to small values of L by the exponential size of the search space; therefore, many heuristic algorithms have been developed. They use some plausible rules to locate good sequences more quickly. A well-known method for such techniques for LABS is *steepest descend local search* (SDLS) [1] or tabu search [6, 8, 11]. In recent years, some modern solvers based on the self-avoiding walk concept have been proposed. The most promising solvers are *lssOrel* [3] and *xLostavka* [4], which are successfully used to find skew-symmetric sequences of lengths of between 301 and 401 with a high merit factor [5].

Another direction of research is using evolutionary multi-agent systems with local optimization algorithms [13]. This hybrid approach is also very promising, as it combines global knowledge of an evolutionary algorithm with exhaustive local search techniques.

In our research, we currently focus on adopting existing local search algorithms (such as SDLS or Tabu) to the GPGPU architecture or building new algorithms that utilize possibilities given by GPGPUs. In [23], we described how to design the SDLS algorithm with a neighborhood of distance 1 (as proposed in [7]) for the GPGPU architecture. In the single iteration of the algorithm, it searches for all sequences that differ by 1 bit from the input sequence and then chooses the best one; each GPU thread changes one bit and evaluates the newly created sequence in a fully parallel way. Moreover, each thread's block of the GPGPU unit computes a different sequence at the same time. Assuming that modern graphics cards provide around 80 thread blocks (represented by hardware multiprocessors), we can achieve the significant improvement in effectiveness that was widely presented in [23].

Another direction of our research is the combination of neural networks with the Tabu search [25]. The aim of this technique is to develop a more efficient method for determining the value of parameter M (the number of iterations for which a selected bit should be blocked), the use of which enables finding a more optimal value for the LABS energy. For this purpose, the LSTM neural network predicts the most effective value of this parameter for a given input sequence S.

In this paper, we propose a new algorithm that is derived from basic SDLS and utilizes the parallel nature of GPGPU. It extends the notion of the sequence neighborhood to a 2-bit distance and allows for the exploration of many more sequences before choosing the best, which minimizes stacking in the local optimum. We also developed the SDLS-DT variant that we proposed in [24], which introduces the recurrent exploration of sequences in both the 1-bit and 2-bit neighborhoods. Both of the new techniques were compared with the basic version of SDLS to verify whether the proposed extensions are more suitable for a parallel computing model. All of them were implemented on the CPU using OpenMP or on GPGPU using CUDA.

The next section provides a brief reminder of the SDLS-DT algorithm that was introduced in [24]. The main part of the paper then presents a new algorithm (called SDLS-2), followed by details about the parallel implementation of all of the referenced algorithms. At the end, the results of experiments using the CPU and GPGPU platform are presented together with conclusions and further work directions.

2. SDLS with deep through search algorithm

The first proposed extension of the SDLS algorithm is *SDLS deep through* (SDLS-DT), which was widely described in [24]. This approach explores the sequence neighborhood in a depth-first manner, but it starts from all sequences that are in the close neighborhood of the input sequence; therefore, it significantly extends the explored solutions space. In this approach, it is not possible to estimate the number of solutions generated in a single step of the algorithm. The searching of solutions in localities one and two is done in a single step in this case. The sequential version of this method is described in Algorithm 1 and is based on *external* and *internal* loops.

Algorithm 1 Sequential version of SDLS-DT algorithm [24]

```
1: function SequntialSDLS-DT($S$)
2:     $E_r = compute\_reference\_energy(S)$
3:     for $i := 0$ to $len(L)$ do
4:         $E[i] = compute\_single\_energy\_by\_mutation\_i_{th}\_bit(S)$
5:         $improvement := true$
6:         while improvement do
7:             $E_{local\_II} = compute\_energies\_by\_mutation\_of\_two\_bits(S)$
8:             $E_{best=}compute\_lowest\_energy(E[i], E_{local\_II})$
9:             if $E_r < E_{best}$ then $improvement := false$
10:            end if
11:            $update\_reference\_energy()$
12:            $update\_reference\_sequence()$
13:        end whileend while
14:    end forend for
15:    return $E_{best}$
16: end functionend function
```

The single step of the described algorithm should be defined as a single run of the external loop. At the beginning of the single run of this loop, the input sequence changes on the position with the index with the same value as the current loop's counter. Based on this sequence, the first energy is calculated (in the first run, this energy is treated as the current reference energy), and this sequence becomes the input to the internal loop. In each iteration of internal loop L, energies are calculated based on the sequences that are different on a two-bit comparison to the original sequence. The lowest of the obtained energies is chosen; in the case where its value is lower than the current reference energy, an update of reference energy E_r and the input sequence corresponding to it occurs. Each single run of the internal loop is performed in those cases in which improvement is not observed (the current reference energy is lower than the best chosen), which is the equivalent of breaking the *while* loop. The number of these steps is equal to the length of the input sequence (the counter of the external loop is equal to L). Figure 1 illustrates the first two steps of the algorithm on a sample sequence.

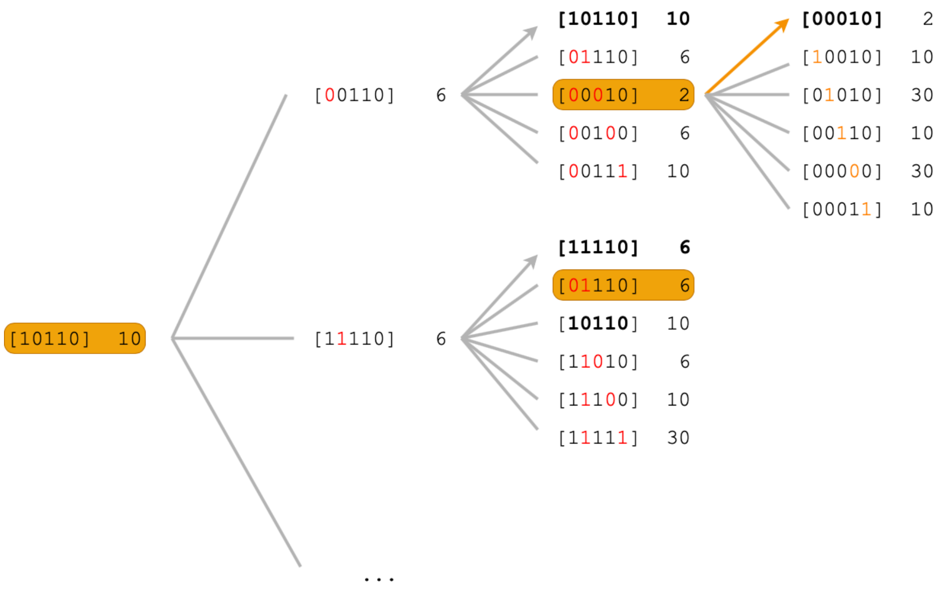


Figure 1. Example of iteration in SDLS-DT algorithm [24]

The parallel version of SDLS-DT for GPGPU is presented in Algorithm 2.

Algorithm 2 Parallel version of SDLS-DT on GPGPU [24]

```
1: function ParallelSDLS-DT($S$)
2:     for $bit := 0$ to $len(L)$ do
3:         if $threadId == bit$ then $S_{block}[threadId] * = -1$ end if
4:         end if
5:         $create\_T(S)\_and\_C(S)()$
6:         $E_r := compute\_energy\_with\_local\_one(S_{block})$
7:         while $improvement$ do
8:             $mutation\_of\_threadId\_bit(S_{block})$
9:             $ParallelValueFlip(S_{block}, T', C')$
10:            $E_{best} := compute\_lowest\_energy()$
11:            if $E_r < E_{best}$ then $improvement := false$ end if
12:            end if
13:            $update\_reference\_energy()$
14:            $update\_reference\_sequence()$
15:            $update\_T(S)\_and\_C(S)()$
16:        end whileend while
17:        if $E_{best_{global}} < E_{best}$ then $E_{best_{global}} := E_{best}$ end if
18:        end if
19:    end forend for
20:    return $E_{best_{global}}$
21: end functionend function
```

In this case, in each iteration of the inner loop L energies are calculated according to the parallel version of the SDLS algorithm [23]. In this implementation, the L energies are calculated using the L threads. This description is about the single-thread block. Similarly to each SDLS version that solves the LABS problem, the K thread's blocks are running at the same time, and the best energy from the K energies is chosen by using the double reduction operation [23].

3. SDLS-2 algorithm with extended neighborhood

The new approach to solving the LABS problem through the use of an innovation algorithm (called *SDLS-2*) relies on increasing the search area of the searching during each single iteration.

One of these is an algorithm that was proposed by Cotta et al. [8], which improves the speed of the computation of the sequence's energy. It introduces the notion of the neighborhood of a sequence S with length L obtained by flipping one symbol in the sequence:

$$N(S) = \{flip(S, i), i \in \{1, .., L\}\} \tag{3}$$

where $flip(s_1 \ldots s_i \ldots s_L, i) = s_1 \ldots s_i \ldots s_L$ [8].

All of the computed products can then be stored in a $(L - 1) \times (L - 1)$ table $T(S)$ such that $T(S)_{ij} = s_j s_{i+j}$ for $j \leq L - i$, and saving the values of the different correlations in a $L - 1$ dimensional vector $C(S)$, which is defined as $C(S)_k = C_k(S)$ for $1 \leq k \leq L - 1$. Figure 2 shows these data structures for an instance of L = 5. Cotta observed that by flipping a single symbol s_i multiple by -1 the value of cells in $T(S)$ where s_i is involved, the fitness of sequence $flip(S, i)$ can be efficiently recomputed in time $O(L)$.

s_1s_2	s_2s_3	s_3s_4	s_4s_5
s_1s_3	s_2s_4	s_3s_5	
s_1s_4	s_2s_5		
s_1s_5			

$T(S)$

$s_1s_2 + s_2s_3 + s_3s_4 + s_4s_5$
$s_1s_3 + s_2s_4 + s_3s_5$
$s_1s_4 + s_2s_5$
s_1s_5

$C(S)$

Figure 2. Data structures supporting efficient computation of fitness in neighborhood

3.1. Sequential version of SDLS-2

The SDLS-2 algorithm (Alg. 3), extends the notion of neighborhood to sequences that differ by up to 2 bits. In this case, in addition to searching for the solution in the neighborhood with a distance equal to 1 (Alg. 3, line 5); the best results in a single iteration are explored among sequences that differ on two bits with regard to the input sequence (Alg. 3, line 6). If the best sequence has a higher energy than the best sequence from the current iteration (Alg. 3, line 8), the last one becomes the reference sequence for the next iterations (Alg. 3, lines 9-10). If the best sequence founded in the current iteration is worse than the input sequence, the algorithm is stopped, and the actual reference sequence is returned as a result. In the case of the sequence with length L, in the single step, this algorithm implies the need to look through $\frac{L(L-1)}{2}$ more solutions than the traditional SDLS algorithm [23].

Algorithm 3 Sequence version of SDLS-2 algorithm

```
1:  function SequntialSDLS-2($S$)
2:      $improvement := true$
3:      $E_r = compute\_reference\_energy(S)$
4:      while improvement do
5:          $E_{local\_I} = calcuate\_energies\_by\_mutation\_of\_single\_bit(S)$
6:          $E_{local\_II} = calcuate\_energies\_by\_mutation\_of\_two\_bits(S)$
7:          $E_{best=}compute\_lowest\_energy(E_{local\_I}, E_{local\_II})$
8:          if $E_r < E_{best}$ then $improvement := false$
9:          end if
10:         $update\_reference\_energy()$
11:         $update\_reference\_sequence()$
12:     end whileend while
13:     return $f$
14: end functionend function
```

The iteration example for the sequence of length $L = 5$ is presented in Figure 3. In this case, the algorithm stops after two iterations.

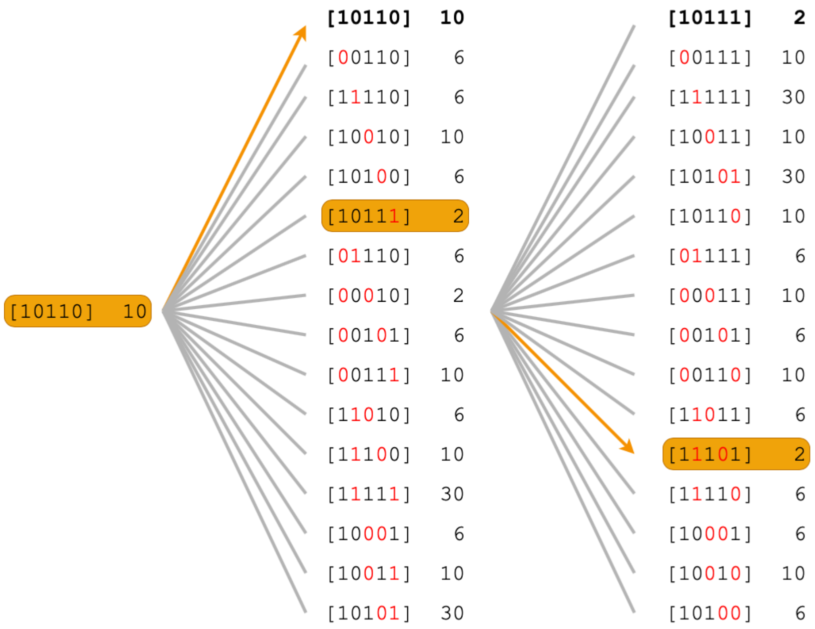


Figure 3. SDLS-2 iterations on sample $L = 5$ sequence

3.2. Parallel version of SDLS-2 for GPGPU

This algorithm can be further parallelized and adjusted to the GPGPU architecture as defined in Algorithm 4. The parallel SDLS algorithm [23] is able to efficiently find the best energy in the neighborhood with the locality one. The moment that this optimal algorithm will be run L times with the inputs where each of them is different on only one other position in regard to the original sequence, the space with locality two will be searched. This assumption forms the basis of the approach that is proposed in this section.

Algorithm 4 Parallel version of SDLS-2 on GPGPU

```
1: function ParallelSDLS-2($S$)
2:     $create\_T(S)\_and\_C(S)()$
3:     $E_r = compute\_reference\_energy(S_{block})$
4:     while $improvement$ do
5:         $mutation\_of\_threadId\_bit(S_{block})$
6:         $ParallelValueFlip(S_{block}, T', C')$
7:         $E_{best} := E_{best\_local1} := compute\_lowest\_energy()$
8:         for $bit := 0$ to $len(L-1)$ do
9:             if $threadId == bit$ then $S_{block}[threadId] * = -1$ end if
10:            end if
11:            $update\_T(S)\_and\_C(S)()$
12:            $mutation\_of\_threadId\_bit(S_{block})$
13:            $ParallelValueFlip(S_{block}, T', C')$
14:            $E_{r_local_2} := compute\_lowest\_energy\_for\_current\_bit()$
15:            if $E_{r\_local2}[bit] < E_{r\_local2}[bit-1]$ then
16:                $E_{best\_local_2} := E_{r\_local2}[bit]$ end if
17:            end if
18:            if $E_{best\_local_2} < E_{best\_local_1}$ then
19:                $update\_C(S)\_optimum\_and\_E\_best$
20:            end ifend if
21:        end forend for
22:        if $E_r < E_{best}$ then $improvement := false$
23:        end if
24:        $update\_reference\_energy()$
25:        $update\_reference\_sequence()$
26:        $update\_T(S)\_and\_C(S)()$
27:    end whileend while
28: end functionend function
```

At the beginning of the SDLS-2 algorithm, there is a searched space with locality one (Alg. 4, lines 2-3). In order not to double-check the same sequence, the search in the space with locality two is realized through the use of the loop with the number of iterations equal to $L-1$ (Alg. 4, line 8). The first step of this iteration is changing one bit in the original sequence to the position with the same index as the actual loop counter and it is realized by the thread with that index (Alg. 4, line 9). This sequence becomes the input sequence for the next steps of the algorithm, so it is necessary to update helper structure $C(S)$ and $T(S)$ (Alg. 4, line 10). This actualization is performed in the same way as it is in the SDLS algorithm.

This created sequence is put as an input to SDLS with the use of $L-1$ threads so that the number of energies is calculated (Alg. 4, lines 11–13), each of which was obtained from the sequences where they were all different at two positions with respect to the original input sequence. The reason why first iteration $L-1$'s energies are calculated with the use of $L-1$ threads instead of calculating L energies that are calculated with the use of L threads is the fact that, during this iteration, thread th_0 is blocked. For example, for the sequence $1,1,1,1,1$, the input for the first iteration is the sequence $0,1,1,1,1$. If thread th_0 calculates the energy in the first iteration, its sequence would be $1,1,1,1,1$. For this sequence, the energy was calculated in the first step of the algorithm (during the calculation of the reference energy) when the energies were obtained through the use of the SDLS algorithm with locality one. For this reason, it makes no sense to calculate this energy again. Of those, the $L-1$ energy is chosen as the one with the lowest value. The best energy is chosen through the use of the double reduction operation, which returns their position in addition to the value of the energy; this enables reproducing the sequence that was used to calculate this minimum energy.

In the second iteration, the input sequence is changed on the position with the index number, which is realized by the thread with that ID. This thread is then turned off from future calculations for the same reason that th_0 was turned off from the calculation during the first iteration. In this iteration, thread th_0 is also not active because, in the current iteration for the same example of the original sequence as in the above example (consisting of only one values), the input sequence is $1,0,1,1,1$. If thread th_0 calculates the energy, then the sequence for it would be in the shape of $0,0,1,1,1$; however, the energy was calculated in the previous iteration by thread th_1 for this sequence. Therefore, in the second iteration, $L-2$ energies are calculated; after this, the one with the lowest value is chosen. This energy is compared with the best energy from the first step of the algorithm, and the better energy is set as the current best energy that was obtained after searching the space with localities one and two (Alg. 4, lines 14–15).

Furthermore, new helper structure $C(S)\_optimum$ was introduced. This structure is needed because, at this stage of running the algorithm, there is no possibility to predict if the best energy will be obtained from the sequence after changing one or two bits (the first or second step of the algorithm). After the first step of the described algorithm, this structure contains the v values that were copied from the cache memory of the winning thread. When the best energy from the second step of the algorithm is better than the one from the first step, structure $C(S)\_optimum$ is updated by the v values of the winning thread from the second step of the best iteration (Alg. 4, line 17). According to the proposed approach, the one-less solution is obtained in each of the next iterations with the use of the one-less thread than in the current iteration.

Having calculated the energies in this way, the best one with the sequence base on which it was calculated is taken. Afterward, this energy is compared with the actual reference energy (in the case of the first iteration, the reference energy is calculated by use of the original input sequence). In those cases in which the value of the best energy is lower than the value of the reference energy, this best energy becomes a new reference energy, its sequence becomes the input sequence to the next iteration, and structures $C(S)$ and $T(S)$ are actualized (Alg. 4, lines 19–21). The algorithm stops searching when the best energy that is found is higher than the actual reference energy. Continuing the running of the algorithm in this condition would cause the same value of best energy to be returned because, according to the algorithm, the input sequence is changed only if, in the current state, it was possible to identify a lower energy than in the previous steps.

The above description concerns the calculations that are realized by the single-thread block. As in the case of the basic SDLS version, K thread blocks are running at the same time. Consequently, as a result of the algorithm, K energies are returned; from these, the best is chosen through the use of a double reduction operation. This energy is treated as a result of a single run of the SDLS-2 algorithm. As mentioned in the single thread block, the algorithm runs until an improvement is achieved in the current step in comparison with the previous one. Each thread block receives a unique input sequence as far as possible (when two thread blocks receive the same input sequence, the result will be the same for both); so, each block should finish its search after a different time. Therefore, the time when all of the algorithms would be completed is equal to the searching time of the longest-running thread block.

4. Parallel version of three variants of SDLS for CPU

The parallel implementation of the CPU of three variants of the SDLS algorithm was performed with the C++ language using the OpenMP[1] library to improve the performance. As a result of the application of OpenMP, the calculations are performed in a parallel manner. By using OpenMP, the threads are created automatically through the use of special directives, which indicate the place on the code that will be run in a parallel manner. In the experiments, a special compiler directive (*-03*) was used to optimize the multicore CPU implementation. The experiments were performed on an Intel Xeon Gold 5220 3.9 GHz that contains 36 cores. In the GPGPU implementation of SDLS, SDLS-2, and SDLS-DT, it was possible to achieve parallelism on number of the solution level (number of the thread's blocks) and during the performing of the SDLS calculation where the L threads are calculating L energies at the same time [23] (in each version of the presented algorithms, the basic version of SDLS is calculated). In the CPU's implementation, it is possible to use only one-level parallelism. As it turns out, the most effective way is to introduce a parallel on the number solution level that is equivalent to the K thread block in the GPGPU implementation.

[1]http://www.openmp.org/

Consequently, *K'* solutions are calculated by *K'* threads at the same time, where *K'* means the number of available cores of the CPU (in our case, this number equals 36).

5. Effectiveness of proposed algorithms

In order to measure the effectiveness of the proposed new SDLS algorithms versus the basic version, each of them sought the optimal solutions for three different input lengths $(128, 192, 256)$ for a period of one hour. The number of solutions was 128. In the first iteration, the processor randomized 128 different sequences (as far as possible) – one for each block in the GPU implementation, and one for each thread in the CPU implementation. With data generated in such a way, each kernel/thread started searching the minimum value of energy. The moment that the best energy from each block/thread was found, it was stored as current best energy $E_{global\_optimum}$. The processor generated a new set of 128 sequences for which the algorithm repeated the search process and, if applicable, the global energy was updated. The entire process was continued for one hour. In addition, the actual best result was stored every two minutes. This experiment for each algorithm was duplicated ten times.

Tables 1 and 2 show the numbers of solutions found together with the averages of the solutions found for the single thread block and the numbers of individual kernels/functions executed for both parallel implementations of GPGPU and CPU.

Table 1
Number of explored solutions during one-minute computations on GPGPU

Method	Number of searched solutions	Average	Standard deviation	Number of running kernels
Sequence length – 128				
SDLS	20,740,434,073	2807	531.8	57,773
SDLS-2	16,865,830,144	224,089	41,837.6	593
SDLS-DT	31 176 794 592	356,752	46,302.4	703
Sequence length – 192				
SDLS	19,152,109,696	5458	1116	27,303
SDLS-2	15,342,863,872	818,216	145,562	148
SDLS-DT	25,827,182,720	1,056,786	121,272	191
Sequence length – 256				
SDLS	22,380,865,408	11,159	2836	15,661
SDLS-2	15,975,437,994	1,657,780	257,819	74
SDLS-DT	24,978,089,130	3,019,130	292,121	68

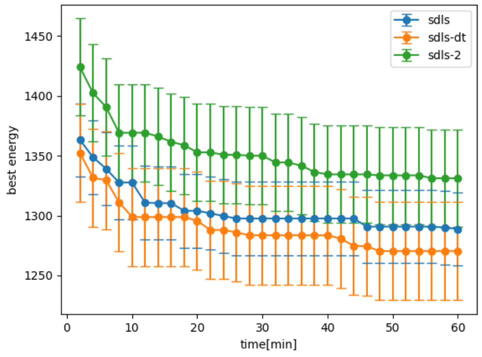
Table 2
Number of explored solutions during one-minute computations on CPU

Method	Number of searched solutions	Number of running functions
Sequence length – 128		
SDLS	470,069,120	6462
SDLS-2	281,504,512	47
SDLS-DT	444,684,288	46

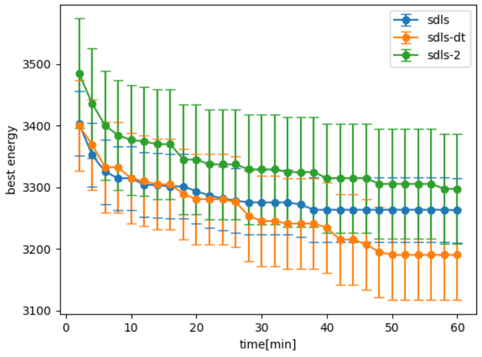
Table 2 cont.

Method	Number of searched solutions	Number of running functions
Sequence length – 128		
SDLS	470,069,120	6462
SDLS-2	281,504,512	47
SDLS-DT	444,684,288	46
Sequence length – 192		
SDLS	467,266,944	3302
SDLS-2	264,717,824	17
SDLS-DT	557,498,304	21
Sequence length – 256		
SDLS	475,191,552	2881
SDLS-2	267,377,536	74
SDLS-DT	490,716,416	10

a)



b)



c)

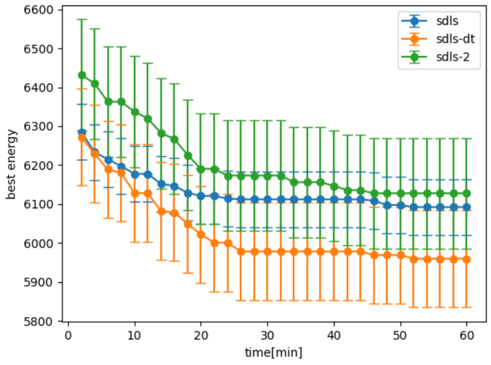


Figure 4. Energy achieved on GPGPU by basic SDLS and SDLS-2, SDLS-DT: a) Energy of LABS $L = 128$; b) Energy of LABS $L = 192$; c) Energy of LABS $L = 256$

Figure 4 shows the efficiency of the presented SDLS variants – all implemented on GPGPU (Figures 4a, 4b, and 4c present the results for LABS 128, 192, and 256 consecutively). The best results in all cases were achieved by the SDLS-DT variant, and the basic SDLS beat SDLS-2; however, the further analysis presented below will explain the differences in more detail. The same configurations were also run using a pure CPU implementation – the results are shown in Figure 5. SDLS-DT still gave the best results, but the SDLS-2 variant now had better convergence than basic SDLS for all problem sizes (Figures 5a, 5b, and 5c show the results for LABS 128, 192, and 256 in succession). Since the SDLS-2 algorithm explored many more sequences in a single run, it may have initially produced worse solutions than basic SDLS; however, all of the plots show that, after exploring a larger neighborhood, it eventually produced better solutions.

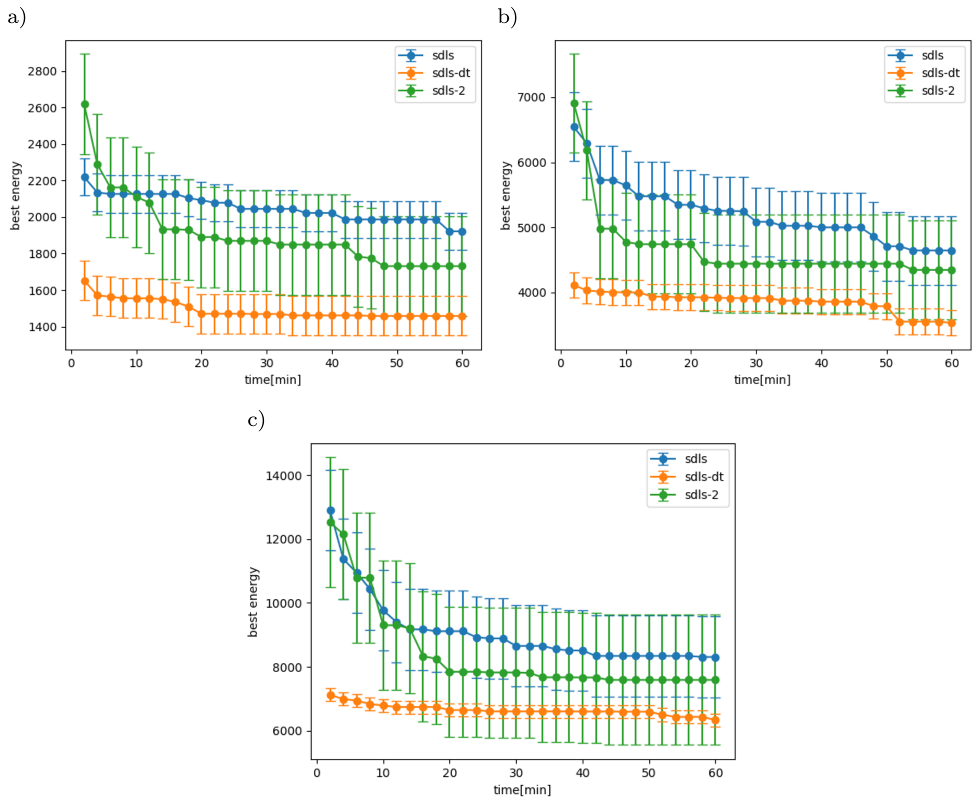


Figure 5. Energy achieved on CPU by basic SDLS and SDLS-2, SDLS-DT: a) Energy of LABS $L = 128$; b) Energy of LABS $L = 192$; c) Energy of LABS $L = 256$

To summarize the results, the best solution was obtained by the *SDLS deep through* algorithm for each problem size; this was due to the efficient parallel implementation of this algorithm. Table 3 compares the size of the search space and the best sequences obtained with the baseline version of the SDLS algorithm. In the case of the SDLS-DT algorithm, it can be seen that the GPU implementation allowed a much larger region of the solution space to be searched; in both versions, the algorithm found better sequences. For one hour, it was able to successively check larger spaces than the SDLS and SDLS-2 algorithms by 50 and 84% for $L = 128$; 34 and 68% for $L = 192$; and 12 and 56% for $L = 256$, respectively.

Table 3
Size of search space explored by new algorithms compared to SDLS, and comparison of best solutions achieved

Problem size	Platform	SDLS-2 search space	SDLS-2 best result	SDLS-DT search space	SDLS-DT best result
128	GPU	81%	97%	150%	10%
192	GPU	80%	98%	135%	102%
256	GPU	71%	99%	112%	101%
128	CPU	60%	110%	95%	125%
192	CPU	57%	106%	119%	124%
256	CPU	56%	109%	103%	124%

Analyzing the table, it is also important to note the interesting behavior of the SDLS-2 algorithm. At the same time, it searched a much smaller number of solutions on both the CPU and GPU; this was due to the inability to use the simplified mechanism for determining the energy of the sequences with direct neighborhoods (differing by one bit) that was used in the other variants of the SDLS algorithm. Nevertheless, the algorithm found sequences comparable to the baseline version of the SDLS algorithm – at 97-99% for the GPU implementation, and at 106-110% for the CPU implementation. Therefore, it can be hypothesized that searching the space in the extended neighborhood (according to the SDLS-2 algorithm) allowed them to find more-efficient paths to search the solution spaces than the other SDLS algorithms. However, the current implementation of the algorithm required more computation in determining the sequence energy. The improvement of this implementation is considered for future work.

6. Conclusions and further work

This paper is the next step of our research related to efficient algorithms for LABS realized using GPGPU architectures. As shown before in [24], the SDLS-DT algorithm can be efficiently implemented on GPGPU and explores more LABS sequences than the other two algorithms, giving the best solutions. On the other hand, the SDLS-2 algorithm explores much fewer sequences, but the quality of the solutions is promising.

The new SDLS algorithms presented show a significant improvement in effectiveness compared to the traditional SDLS approach. Implementation can be further integrated with meta-heuristics such as evolutionary algorithms, which constitute a basis for the concept of a hybrid computational environment in the master-slave model that was proposed in [17].

In the near future, the authors plan to implement parallel variants of the self-avoiding walk algorithm or other solvers such as *lssOrel* or *xLostavka.* We will also show that we considered extending the search neighborhood in tabu search heuristics and propose new variants of the algorithm dedicated to the GPGPU. The next interesting and promising direction of research is to design and implement an evolutionary multi-agent system with local optimization on GPGPU units.

Acknowledgements

The research presented in this paper was realized thanks to the funds of the Polish Ministry of Science and Higher Education assigned to AGH University of Science and Technology.

References

[1] Bartholomew-Biggs M.: The Steepest Descent Method. In: *Nonlinear Optimization with Engineering Applications*, pp. 1–8, Springer US, Boston, MA, 2008. doi: 10.1007/978-0-387-78723-7_7.

[2] Bernasconi J.: Low autocorrelation binary sequences: statistical mechanics and configuration space analysis, *Journal De Physique*, vol. 48, pp. 559–567, 1987. doi: 10.1051/jphys:01987004804055900.

[3] Bošković B., Brglez F., Brest J.: Low-Autocorrelation Binary Sequences: On Improved Merit Factors and Runtime Predictions to Achieve Them, *arXiv e-prints*, arXiv:1406.5301, 2014. doi: 10.1016/j.asoc.2017.02.024. 1406.5301.

[4] Brest J., Bošković B.: A Heuristic Algorithm for a Low Autocorrelation Binary Sequence Problem With Odd Length and High Merit Factor, *IEEE Access*, vol. 6, pp. 4127–4134, 2018. doi: 10.1109/ACCESS.2018.2789916.

[5] Brest J., Bošković B.: In Searching of Long Skew-symmetric Binary Sequences with High Merit Factors, 2020.

[6] Dotú I., Van Hentenryck P.: A Note on Low Autocorrelation Binary Sequences. In: F. Benhamou (ed.), *Principles and Practice of Constraint Programming – CP 2006*, pp. 685–689, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi: 10.1007/11889205_51.

[7] Gallardo J.E., Cotta C., Fernandez A.J.: A Memetic Algorithm for the Low Autocorrelation Binary Sequence Problem. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 1226–1233, GECCO '07, ACM, New York, NY, USA, 2007. doi: 10.1145/1276958.1277195.

[8] Gallardo J.E., Cotta C., Fernández A.J.: Finding Low Autocorrelation Binary Sequences with Memetic Algorithms, *Appl Soft Comput*, vol. 9(4), pp. 1252–1262, 2009. doi: 10.1016/j.asoc.2009.03.005.

[9] Golay M.: The merit factor of long low autocorrelation binary sequences (Corresp.), *IEEE Transactions on Information Theory*, vol. 28(3), pp. 543–549, 1982. doi: 10.1109/tit.1982.1056505.

[10] Günther C., Schmidt K.U.: Merit factors of polynomials derived from difference sets, 2016. doi: 10.1016/j.jcta.2016.08.006.

[11] Halim S., Yap R., Halim F.: Engineering Stochastic Local Search for the Low Autocorrelation Binary Sequence Problem. In: *Principles and Practice of Constraint Programming*, vol. 5202, pp. 640–645, 2008. doi: 10.1007/978-3-540-85958-1_57.

[12] Jedwab J., Katz D.J., Schmidt K.U.: Advances in the merit factor problem for binary sequences, *Journal of Combinatorial Theory, Series A*, vol. 120(4), pp. 882–906, 2013. doi: 10.1016/j.jcta.2013.01.010.

[13] Kowol M., Byrski A., Kisiel-Dorohinicki M.: Agent-based Evolutionary Computing for Difficult Discrete Problems, *Procedia Computer Science*, vol. 29, pp. 1039–1047, 2014. doi: 10.1016/j.procs.2014.05.093.

[14] Leukhin A.N., Potekhin E.N.: A Bernasconi model for constructing ground-state spin systems and optimal binary sequences, *Journal of Physics: Conference Series*, vol. 613, p. 012006, 2015. doi: 10.1088/1742-6596/613/1/012006.

[15] Moore C., Mertens S.: *The Nature of Computation*, Oxford University Press, Inc., USA, 2011. doi: 10.1093/acprof:oso/9780199233212.001.0001.

[16] Packebusch T., Mertens S.: Low autocorrelation binary sequences, *Journal of Physics A: Mathematical and Theoretical*, vol. 49(16), p. 165001, 2016. doi: 10.1088/1751-8113/49/16/165001.

[17] Piętak K., Żurek D., Pietroń M., Dymara A., Kisiel-Dorohinicki M.: Striving for performance of discrete optimisation via memetic agent-based systems in a hybrid CPU/GPU environment, *Journal of Computational Science*, vol. 31, pp. 151–162, 2019. doi: https://doi.org/10.1016/j.jocs.2019.01.007.

[18] Ukil A.: On asymptotic merit factor of low autocorrelation binary sequences. In: *IECON 2015 – 41st Annual Conference of the IEEE Industrial Electronics Society*, pp. 004738–004741, 2015. doi: 10.1109/IECON.2015.7392840.

[19] Velazquez-Gutierrez J.M., Vargas-Rosales C.: Sequence Sets in Wireless Communication Systems: A Survey, *IEEE Communications Surveys Tutorials*, vol. 19(2), pp. 1225–1248, 2017. doi: 10.1109/COMST.2016.2639739.

[20] Walsh T.: CSPLib Problem 005: Low Autocorrelation Binary Sequences, http://www.csplib.org/Problems/prob005. Accessed: 2017-01-31.

[21] Zeng F., He X., Zhang Z., Xuan G., Peng Y., Yan L.: Optimal and Z-Optimal Type-II Odd-Length Binary Z-Complementary Pairs, *IEEE Communications Letters*, vol. 24(6), pp. 1163–1167, 2020. doi: 10.1109/LCOMM.2020.2977897.

[22] Zhao L., Song J., Babu P., Palomar D.P.: A Unified Framework for Low Autocorrelation Sequence Design via Majorization–Minimization, *IEEE Transactions on Signal Processing*, vol. 65(2), pp. 438–453, 2017. doi: 10.1109/TSP.2016.2620113.

[23] Żurek D., Piętak K., Pietroń M., Kisiel-Dorohinicki M.: Toward hybrid platform for evolutionary computations of hard discrete problems, *Procedia Computer Science*, vol. 108, International Conference on Computational Science, ICCS 2017, 12–14 June 2017, Zurich, Switzerland, pp. 877–886, 2017. doi: 10.1016/j.procs.2017.05.201.

[24] Żurek D., Piętak K., Pietroń M., Kisiel-Dorohinicki M.: New Variants of SDLS Algorithm for LABS Problem Dedicated to GPGPU Architectures. In: M. Paszyński, D. Kranzlmüller, V.V. Krzhizhanovskaya, J.J. Dongarra, P.M.A. Sloot (eds.), *Computational Science – ICCS 2021*, pp. 206–212, Springer International Publishing, Cham, 2021. doi: 10.1007/978-3-030-77961-0_18.

[25] Żurek D., Pietroń M., Piętak K., Kisiel-Dorohinicki M.: A Deep Neural Network as a TABU Support in Solving LABS Problem. In: D. Groen, C. de Mulatier, M. Paszynski, V.V. Krzhizhanovskaya, J.J. Dongarra, P.M.A. Sloot (eds.), *Computational Science – ICCS 2022*, pp. 237–243, Springer International Publishing, Cham, 2022. doi: 10.1007/978-3-031-08754-7_32.

Affiliations

Dominik Żurek
AGH University of Krakow, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Kamil Piętak
AGH University of Krakow, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Marcin Pietroń
AGH University of Krakow, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Marek Kisiel-Dorohinicki
AGH University of Krakow, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Received: 21.10.2024
Revised: 02.02.2025
Accepted: 04.05.2025