A Parallel Simulated Annealing Algorithm for Weapon-Target Assignment Problem

Emrullah SONUC
Department of Computer Engineering
Karabuk University
Karabuk, TURKEY

Baha SEN
Department of Computer Engineering
Yildirim Beyazit University
Ankara, TURKEY

Safak BAYIR
Department of Computer Engineering
Karabuk University
Karabuk, TURKEY

***Abstract*—Weapon-target assignment (WTA) is a combinatorial optimization problem and is known to be NP-complete. The WTA aims to best assignment of weapons to targets to minimize the total expected value of the surviving targets. Exact methods can solve only small-size problems in a reasonable time. Although many heuristic methods have been studied for the WTA in the literature, a few parallel methods have been proposed. This paper presents parallel simulated algorithm (PSA) to solve the WTA. The PSA runs on GPU using CUDA platform. Multi-start technique is used in PSA to improve quality of solutions. 12 problem instances (up to 200 weapons and 200 targets) generated randomly are used to test the effectiveness of the PSA. Computational experiments show that the PSA outperforms SA on average and runs up to 250x faster than a single-core CPU.**

Keywords—Weapon-Target Assignment; Multi-start Simulated Annealing; Combinatorial optimization; Parallel algorithms; GPU

I. Introduction

The Weapon-Target Assignment (WTA) problem is an NP-complete combinatorial optimization problem at field of military operation research [1]. The WTA Problem aims to find best assignment of weapons to targets, to minimize the expected damage of the defended area in order to increase chances of survival. Several exact methods are studied in the literature [2-4] but these methods can solve only small-size problems. Thus, heuristic methods such as Simulated Annealing [5, 6], Genetic Algorithm [6, 7], Tabu Search [6], Variable Neighborhood Search [3, 6], Ant Colony [7-9] and Particle Swarm Optimization [10] are proposed for the WTA.

Simulated Annealing (SA) is an efficient algorithm for solving the WTA problem [5, 6]. The SA is a flexible algorithm to implement any problem like the WTA. On the other hand, each iteration of the SA depends on the previous iteration. Therefore, runtime of the SA method is not as good enough as other heuristic methods. Parallelization of the SA can be presented as a solution to overcome this problem.

Nowadays, GPUs are very efficient hardware platform to develop parallel algorithms. Several parallel implementations of the SA on GPU are presented in the literature [11-12]. These methods have achieved good quality results in the applied areas. In this paper, a parallel SA algorithm (PSA) is developed to solve the WTA problem. PSA has been developed on GPU and has used the multi-start technique to obtain better results.

This work was supported by the Scientific Research Coordination Unit of Karabuk University under Grant KBU-BAP-15/2-DR-027.

This paper is organized as follows. In Section II, mathematical formulation and definition of the WTA problem is introduced. The SA algorithm is described in Section III. Section IV gives details about the PSA. Computational experiments and results are presented in Section V and finally Section VI states some conclusions.

II. The WTA Problem

In the WTA problem, assets of the defense want to destroy attacks of the targets directed by offense. The defense has a finite number of weapons to defend incoming threats from the offense. There are two models as static and dynamic of the WTA problem. In this paper, the static of the WTA problem has been studied. In the static model, all inputs of the problem are static and the assignments of weapons to targets are performed in a single step. The expected damage for assignments is evaluated after all weapon-target engagements have been completed. Parameters and variables of the problem are defined as follow:

- n , the number of the targets (1, 2, …, n),
- m , the number of the weapons (1, 2, …, m),
- v_i , the value of the target i,
- p_{ij} , the probability of destroying by assigning the jth weapon to the ith target,
- $x = [x_{ij}]$, the decision variable that is nxm matrix, where

$$x_{ij} = \begin{cases} 1 \text{ if weapon } j \text{ is assigned to target } i, \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

The survival probability of target i when attacks weapon j is $\left(1-p_{ij}\right)^{x_{ij}}$. The problem can be formulated as follows:

$$f(\pi) = \sum_{i=1}^{n} v_i \prod_{j=1}^{m} \left(1-p_{ij}\right)^{x_{ij}} \tag{2}$$

$$s.\,t. \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,2,...,m. \tag{3}$$

All weapons must be assigned to targets. In this paper, it is assumed that number of target equals to number of weapons and only one weapon can be assigned to one target.

III. SIMULATED ANNEALING ALGORITHM FOR THE WTA

Simulated Annealing (SA) is a heuristic algorithm to obtain optimum or near-optimum of a given function in a large space. Kirkpatrick and Vecchi [13] have developed the SA in 1983 to solve a problem for economic activities. In the SA algorithm, each step generates a random solution using the current solution that is achieved in the previous step. Acceptance of the new solution depends on parameters of the method and the difference between neighbor solutions. Metropolis criterion [14] and Boltzmann distribution are used for the acceptance of the new solution. Also, these methods ensure that the SA does not stick to a local minimum or maximum. The acceptance probability function is shown as follows:

$$P(\Delta f)=\begin{cases}1 & \text{if } \Delta f<0,\\ exp(-\Delta f/T) & \text{otherwise,}\end{cases} \tag{4}$$

where T is temperature at each step and decreasing by cooling factor (α) for each step. P is acceptation probability of the current solution in annealing process. A new candidate solution is found by randomly selecting two weapons q and r, then swapping assignments to targets between selected weapons. Δf is the difference between two neighbors' solutions in a given function and defined as follows:

$$\Delta f = v_q(p_{qq}-p_{rq})+v_r(p_{rr}-p_{qr}) \tag{5}$$

After swapping operation, the new candidate solution is calculated by formula as given below:

$$f(\pi') = f(\pi) + \Delta f \tag{6}$$

When T reaches to a temperature T_{final} that is determined as a parameter by user, the method is terminated. Except for that, time dependent and iteration number dependent termination methods are also used. The pseudocode of the SA is as follows:

Algorithm 1: Pseudocode of the SA Algorithm.

```
Begin
$T \leftarrow T_0$;
$s \leftarrow s_0$;
$f \leftarrow F(s)$;
$s_{best} \leftarrow s$;
$f_{best} \leftarrow f$;
while $T > T_{final}$ and stopping criterion is not met yet do
        $s_{new} \leftarrow rnd(s)$;
        $f_{new} \leftarrow F(s_{new})$;
        if $P(f, f_{new}, T) > rnd(0,1)$ then
                $s \leftarrow s_{new}$;
                $f \leftarrow f_{new}$;
        end if
        if $f < f_{best}$ then
                $s_{best} \leftarrow s_{new}$;
                $f_{best} \leftarrow f_{new}$;
        end if
$T \leftarrow T \times a$;
end while
End.
```

The function *rnd(s)* returns a random number sequence using simple local search algorithms like swapping, 2-opt etc. for permutation *s* and the function *rnd(0,1)* returns a random number between 0 and 1.

The main steps that compose the SA algorithm for the WTA problem are described below.

Stage 1: Initialization

1) Inputs: Probability of destroying matrix p, value of targets v, array of permutation s of the weapons that is assigned to each target
2) Set the SA parameters: T,T_{final} and a.
3) Solve the WTA using (2) as an initial solution f with the permutation array s that is generated randomly.

Stage 2: The SA Execution

1) Generate two different index numbers randomly for s and swap them.
2) Calculate Δf using (5).
3) Accept or not to accept the s_{new} using $P(\Delta f)$.
4) If the s_{new} is accepted then set $s = s_{new}$ and go to step 8, otherwise go to step 9.
5) Calculate the f_{new} using (6) and set $f = f_{new}$.
6) Set $f_{best} = f_{new}$ and $s_{best} = s_{new}$ if $f < f_{best}$.
7) Set T = T . a;
8) Repeat Step 4 – Step11 until T is reached to T_{final}.

In the above stages, Stage 2 performs the SA algorithm after initialization of required variables and parameters in Stage 1. Stage 2 searches a new solution by swapping between weapon assignments of two targets (see Fig.1).

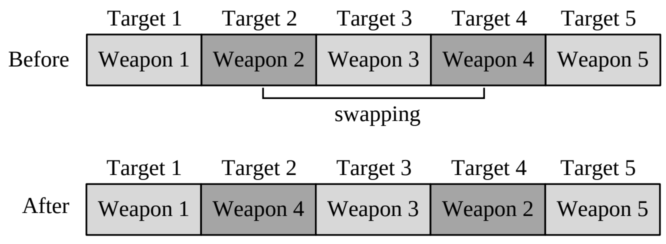

Fig. 1. Swapping between weapon assignments of the targets

IV. PARALELLIZATION ON GPU

Implementation of the PSA has been performed using Compute Unified Device Architecture (CUDA) on Graphics Processing Units (GPUs). CUDA is a C/C++ language extension and a parallel computing platform created by NVIDIA Corporation [15]. CUDA platform is also a tool for General Purpose Computing on Graphics Processing Units (GPGPUs). GPGPU can be defined as a parallel processing methodology using GPUs for high-performance computing.

The technique of restarting a heuristic algorithm with different configurations is called multi-start and it is an effective method to improve quality of solutions for optimization problems. [16]. This technique is also used for the SA and proved its effectiveness [17, 18]. Only one heuristic method is run at a time and the method must be restarted to use the multi-start technique for a single-core CPUs. On the other hand, several heuristic methods run at the same time on a multi-core CPUs and many-core GPUs. In this paper, the PSA has been proposed with a multi-start technique.

In the PSA, every thread on GPU starts with a different *s*. cuRAND is a pseudorandom number generator library defined on CUDA platform and is used to provide multi-start technique. All threads have a different seed and different seeds are guaranteed to produce different sequences. Each thread runs the SA independently that is given the steps at Stage 2 in Section III. After each thread has run the SA, threads in a same block communicate with each other using shared memory. The best fitness value is found for each block in parallel using reduction method. The flowchart of the PSA handled by each thread is shown in Fig. 2.

After the best fitness values have been found for each block of the GPU, they are transferred to CPU. These operations are shown at the process before the Stop process in Fig. 2. Finding the best fitness value has been operated in parallel using reduction method. In the reduction method, half of the threads on a block are active. Transfer of the best fitness value of each block is performed by the first threads on blocks. In other words, only first thread on each block is active for transferring. After that, the best fitness value is found on CPU from the best values of all blocks.

In this paper, 1024 threads per block on GPU have been used. This means all threads on a block are used. When the number of blocks is increased, runtime of the PSA increases, too. The reason of this increment is accessing the global memory at a same time from many threads. Several configurations have been realized to optimize the runtime of the PSA. These are given below.

- *Short* gives better performance than *int* therefore *short* variables are used instead of *int* variables if suitable.
- --use_fast_math is a compiler parameter. It can be chosen for faster mathematical functions.
- Accessing the global memory is very slow in the CUDA platform. Shared memory is used for accessing *v* (values of the targets) to read/write much faster. Global memory must be used for the *p* (matrix array stores probability of destroying targets) because of the limitation of the shared memory.

Each thread performs the swapping operation on its own permutation list. In this context, each thread requires its own a copy of:

- The array of permutation,
- Two integer variables generated by randomly in each step for swapping operation,
- The fitness value of a given function,
- The delta value (difference between current and candidate solutions),
- The temperature value (parameter of the SA).

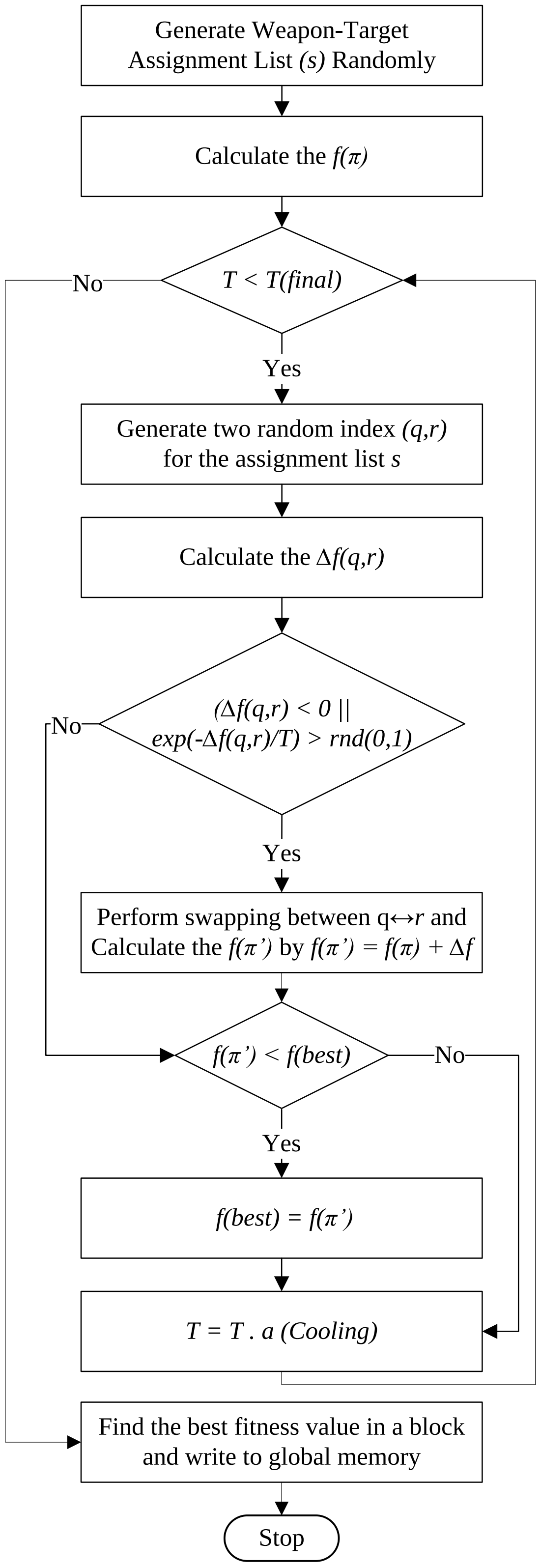
Generate Weapon-Target Assignment List (s) Randomly
Calculate the f(π)
T < T(final)
No
Yes
Generate two random index (q,r) for the assignment list s
Calculate the Δf(q,r)
(Δf(q,r) < 0 || exp(-Δf(q,r)/T) > rnd(0,1))
No
Yes
Perform swapping between q↔r and Calculate the f(π') by f(π') = f(π) + Δf
f(π') < f(best)
No
Yes
f(best) = f(π')
T = T . a (Cooling)
Find the best fitness value in a block and write to global memory
Stop

Fig. 2. Flowchart of the PSA

V. Computational Experiments

There are no benchmark problem datasets/instances in the literature for the WTA problem. That is why, various scenarios created to test the performance for proposed methods. In this paper, computational tests have been carried out on 12 problem instances in different dimensions (available at: *http://web.karabuk.edu.tr/emrullahsonuc/wta*). The values of targets are generated as random numbers from the uniform distribution in the range 25–100. The probabilities of destroying targets for weapon-target assignments are generated as random numbers from the uniform distribution in the range 0.60–0.90. Problem instances are generated with different dimensions which are in the range 5–200. Dimensions of the problem instances (WTA1-WTA12) are shown in Table I.

TABLE. I. Dimensions of the Problem Instances for the WTA

Problem	Weapon	Target
WTA1	5	5
WTA2	10	10
WTA3	20	20
WTA4	30	30
WTA5	40	40
WTA6	50	50
WTA7	60	60
WTA8	70	70
WTA9	80	80
WTA10	90	90
WTA11	100	100
WTA12	200	200

The performance of the SA depends on the parameters belonging to the method. Parameter configurations are determined as $T = 1000$, $T_{final} = 0.1$, $\alpha = 0.99999$. For each problem, results have been obtained by averaging of 10 independent runs on CPU. All tests are performed on CPU with Intel Xeon 2.4 GHz. The results of the SA on problem instances are presented in Table II. The best, the worst, the mean, the median and the standard deviation (SD) are listed in Table II for all problem instances.

TABLE. II. Results of the SA on Problem Instances

Problem	Best	Worst	Mean	Median	SD
WTA1	48.3640	48.3640	48.3640	48.3640	0.00
WTA2	96.3123	96.3123	96.3123	96.3123	0.00
WTA3	142.1070	142.1070	142.1070	142.1070	0.00
WTA4	248.0285	248.0285	248.0285	248.0285	0.00
WTA5	305.5016	305.5016	305.5016	305.5016	0.00
WTA6	353.0767	353.5702	353.3112	353.2610	0.14
WTA7	415.0528	415.7079	415.4068	415.4371	0.21
WTA8	498.1049	499.0167	498.5918	498.5860	0.30
WTA9	534.4408	536.2618	535.4559	535.5937	0.57
WTA10	594.0639	596.1228	595.3277	595.6466	0.72
WTA11	699.8357	702.1189	701.0054	701.2495	0.75
WTA12	1306.9126	1309.4616	1308.3382	1308.5187	0.86

NVIDIA GeForce GTX Titan X (3072 cores, 1.0 GHz) has been used for running the PSA. Results of the PSA for each problem instances are always same. The reason of this result is that the random number sequence generated by each thread is the same at every run. The PSA runs on GPU three times for each problem instance to obtain the average runtime. 24 blocks have been used on GPU to perform the performance comparison with CPU. Table III represents the experimental results of the PSA on problem instances. Table III also shows the best and the mean of the SA (SA Best & SA Mean) for comparing results. The best results are shown in bold.

TABLE. III. Results of the PSA And the SA on Problem Instances

Problem	PSA	SA Best	SA Mean
WTA1	**48.3640**	**48.3640**	**48.3640**
WTA2	**96.3123**	**96.3123**	**96.3123**
WTA3	**142.1070**	**142.1070**	**142.1070**
WTA4	**248.0285**	**248.0285**	**248.0285**
WTA5	**305.5016**	**305.5016**	**305.5016**
WTA6	353.4801	**353.0767**	353.3112
WTA7	**414.7340**	415.0528	415.4068
WTA8	**497.2972**	498.1049	498.5918
WTA9	535.5422	**534.4408**	535.4559
WTA10	595.3730	**594.0639**	595.3277
WTA11	**699.4143**	699.8357	701.0054
WTA12	1307.0154	**1306.9126**	1308.3382

Table III shows that the SA has better accuracy in 4 of 12 problems according to best results. If the mean results are considered, the SA has not any better accuracy than the PSA. The PSA has better accuracy in 3 of 12 problems. The SA and the PSA have same accuracy for 5 of 12 problems. The results maybe the optimum fitness value for these 5 problem instances. For each problem instance, runtime results in seconds and speedups are given in Table IV. Speedup is calculated according to a formula as below:

$$Speedup = \frac{runtime_{SA}}{runtime_{PSA}} \qquad (7)$$

TABLE. IV. Runtimes in Seconds (S) and Speedups

Problem	SA (s)	PSA (s)	Speedup
WTA1	2985.92	19.28	155x
WTA2	2841.04	30.94	92x
WTA3	2752.49	17.56	157x
WTA4	2754.31	11.23	245x
WTA5	2760.78	12.15	227x
WTA6	2790.03	11.17	250x
WTA7	2787.45	14.09	198x
WTA8	2841.02	15.73	181x
WTA9	2868.79	18.12	158x
WTA10	2812.57	19.90	141x
WTA11	2805.83	20.60	136x
WTA12	2902.15	27.57	105x
Average	*2985.92*	*19.28*	*155x*

According to runtime results of the SA for problem instances, runtimes are close to each other on instances and average time is 2985.92 seconds. The average runtime of the PSA is 19.28 seconds. Speedups have been shown also in Fig.3. The average speedup of 12 problem instances is 155x. It can be said that this acceleration is capable of making the SA algorithm more efficient.

The reason why the speedup values do not increase linearly is due to the access on the global memory. Accessing global memory is efficient if there is a coalesced access to it. On the PSA, there is no coalesced accessing in the process of writing best results for each block to global memory. Thus, speedups can be uphill or downhill for various dimensions (see Fig. 3). The best speedup is 250x for the WTA6 problem instance and the worst speedup is 92x for the WTA2 problem instance.

If the PSA is considered, increasing the number of blocks will cause more access to global memory. For this reason, this will also increase the runtime as mentioned before. On the other hand, running of the PSA by more threads means new multi-start configurations. More multi-starts provide the possibility of increasing the quality of the results. The results obtained using 24, 48, 96, 128, 512 and 1024 blocks are shown in Table V. The best results are shown in bold. If the number of blocks is increased, then the quality of the results is improved. Results of first five problem instances (WTA1-WTA5) are same for all runs. When 1024 blocks are used for the PSA, all results have been improved except first five problem instances. Furthermore, when 1024 blocks are used for the PSA, runtime is less than half of the SA method corresponding to the PSA using 24 blocks.

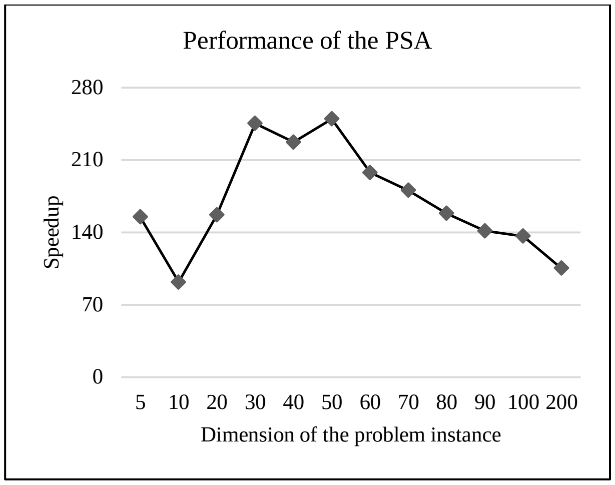

Fig. 3. Speedup against CPU for problem instances

TABLE. V. RESULTS OF THE PSA ON PROBLEM INSTANCES FOR DIFFERENT NUMBER OF BLOCKS

Problem	**Number of Blocks (Results are shown on the first column and runtimes in seconds are shown on the second column)**											
	24		***48***		***96***		***128***		***512***		***1024***	
WTA1	**48.3640**	19.28	**48.3640**	38.49	**48.3640**	75.96	**48.3640**	75.96	**48.3640**	383.39	**48.3640**	764.63
WTA2	**96.3123**	30.94	**96.3123**	59.76	**96.3123**	119.16	**96.3123**	119.16	**96.3123**	623.34	**96.3123**	1245.74
WTA3	**142.1070**	17.56	**142.1070**	33.25	**142.1070**	65.78	**142.1070**	65.78	**142.1070**	324.42	**142.1070**	642.34
WTA4	**248.0285**	11.23	**248.0285**	21.24	**248.0285**	41.66	**248.0285**	41.66	**248.0285**	217.12	**248.0285**	431.41
WTA5	**305.5016**	12.15	**305.5016**	23.18	**305.5016**	45.07	**305.5016**	45.07	**305.5016**	198.64	**305.5016**	391.05
WTA6	353.4801	11.17	353.3609	21.15	**353.0102**	41.03	**353.0102**	41.03	**353.0102**	233.08	**353.0102**	459.42
WTA7	414.7340	14.09	414.7340	26.85	414.7340	52.08	414.7340	52.08	**414.6011**	232.36	**414.6011**	428.31
WTA8	497.2972	15.73	497.2972	30.01	497.2972	58.23	497.2972	58.23	**496.7948**	278.67	**496.7948**	536.04
WTA9	535.5422	18.12	534.6199	34.76	534.3872	68.47	534.3872	68.47	**533.9201**	343.87	**533.9201**	670.37
WTA10	595.3730	19.90	594.4800	38.66	594.4658	76.44	594.4658	76.44	593.6951	382.01	**592.7965**	749.48
WTA11	699.4143	20.60	699.4143	40.25	699.4143	79.31	699.4143	79.31	**697.6242**	401.19	**697.6242**	787.51
WTA12	1307.0154	27.57	1306.8689	55.00	1304.4195	110.80	1304.4195	147.11	1304.4195	586.39	**1302.9348**	1181.51

VI. CONCLUSIONS

In this paper, the PSA is proposed to solve the WTA problem. Multi-start technique is used in the PSA to obtain better results on problem instances. The PSA runs on a GPU using CUDA platform. Results are compared both quality and acceleration. The PSA is up to 250x faster than a single-core CPU. In terms of quality solutions, the PSA is capable of delivering good quality results than the SA for problem instances in average. In future, the PSA can be optimized for coalesced accessing to improve runtime. Also, the PSA can be applied dynamic WTA problem which is other model of the WTA.

REFERENCES

[1] S. P. Lloyd and H. S. Witsenhausen, "Weapons allocation is NP-complete," in 1986 Summer Computer Simulation Conference, 1986, pp. 1054-1058.

[2] F. Ma, M. Ni, B. Gao, and Z. Yu, "An efficient algorithm for the weapon target assignment problem," in Information and Automation, 2015 IEEE International Conference on, 2015, pp. 2093-2097.

[3] R. K. Ahuja, A. Kumar, K. C. Jha, and J. B. Orlin, "Exact and heuristic algorithms for the weapon-target assignment problem," Operations Research, vol. 55, pp. 1136-1146, 2007.

[4] T. Sikanen, "Solving weapon target assignment problem with dynamic programming," Technical report, Mat− 2.4108 Independent research projects in applied mathematics, 2008.

[5] A. M. Madni and M. Andrecut, "Efficient heuristic approach to the weapon-target assignment problem," Journal of Aerospace Computing, Information, and Communication, vol. 6, pp. 405-414, 2009.

[6] A. Tokgöz and S. Bulkan, "Weapon target assignment with combinatorial optimization techniques," IJARAI) International Journal of Advanced Research in Artificial Intelligence, vol. 2, 2013.

[7] J. Zhang, X. Wang, C. Xu, and D. Yuan, "ACGA algorithm of solving weapon-target assignment problem," Open Journal of Applied Sciences, vol. 2, p. 74, 2013.

[8] Z.-J. Lee, C.-Y. Lee, and S.-F. Su, "An immunity-based ant colony optimization algorithm for solving weapon–target assignment problem," Applied Soft Computing, vol. 2, pp. 39-47, 2002.

[9] G. Shang, "Solving weapon-target assignment problems by a new ant colony algorithm," in Computational Intelligence and Design, 2008. ISCID'08. International Symposium on, 2008, pp. 221-224.

[10] X. Zeng, Y. Zhu, L. Nan, K. Hu, B. Niu, and X. He, "Solving weapon-target assignment problem using discrete particle swarm optimization," in Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on, 2006, pp. 3562-3565.

[11] A. Ferreiro, J. García, J. G. López-Salas, and C. Vázquez, "An efficient implementation of parallel simulated annealing algorithm in GPUs," Journal of Global Optimization, vol. 57, pp. 863-890, 2013.

[12] E. Sonuc, B. Sen, and S. Bayir, "A Parallel Approach for Solving 0/1 Knapsack Problem using Simulated Annealing Algorithm on CUDA

Platform," International Journal of Computer Science and Information Security, vol. 14, p. 1096, 2016.

[13] S. Kirkpatrick and M. P. Vecchi, "Optimization by simulated annealing," science, vol. 220, pp. 671-680, 1983.

[14] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," The journal of chemical physics, vol. 21, pp. 1087-1092, 1953.

[15] Nvidia. (2017). "CUDA C Programming Guide". Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[16] R. Martí, M. G. Resende, and C. C. Ribeiro, "Multi-start methods for combinatorial optimization," European Journal of Operational Research, vol. 226, pp. 1-8, 2013.

[17] F. Y. Vincent and S.-W. Lin, "Multi-start simulated annealing heuristic for the location routing problem with simultaneous pickup and delivery," Applied Soft Computing, vol. 24, pp. 284-290, 2014.

[18] S.-W. Lin, "Solving the team orienteering problem using effective multi-start simulated annealing," Applied Soft Computing, vol. 13, pp. 1064-1073, 2013.