# A Parallel Approach for Solving 0/1 Knapsack Problem using Simulated Annealing Algorithm on CUDA Platform

Emrullah SONUC
Department of Computer Engineering
Karabuk University
Karabuk, TURKEY
esonuc@karabuk.edu.tr

Baha SEN
Department of Computer Engineering
Yildirim Beyazit University
Ankara, TURKEY
bsen@ybu.edu.tr

Safak BAYIR
Department of Computer Engineering
Karabuk University
Karabuk, TURKEY
safakbayir@karabuk.edu.tr

***Abstract*— CUDA is a powerful platform and has a high compute capability for developing parallel algorithms. Several parallel heuristic methods on GPUs for combinatorial optimization problems have been studied in the literature. In this paper, 0/1 Knapsack Problem that is one of the most studied combinatorial optimization problem is discussed. An effective simulated annealing based method has been proposed and it runs on CPU and GPU. The proposed method runs in parallel on GPU with multi-start technique to improve quality of solutions. A set of 10 low-dimensional knapsack problems and randomly generated medium-dimensional test instances with up to 1,000 objects are used to test the effectiveness of the method. Computational results on problem instances show that the proposed method with parallel approach outperforms sequential one on average and runs up to 16x faster than a single-core CPU.**

***Keywords- Combinatorial optimization, 0/1 Knapsack Problem, Multi-start Simulated Annealing, Parallel algorithms, CUDA.***

## I. INTRODUCTION

The 0/1 Knapsack Problem (KP) is an NP-hard combinatorial optimization problem. Several exact methods have been proposed for solving KP in the literature. Generally, dynamic programming and branch bound [1, 2] are used to find exact optimal solutions. Other methods are heuristic methods which provide the optimal or near-optimal solutions in a reasonable time. Many heuristic methods have been proposed for the 0/1 KP such as Simulated Annealing [3], Ant Colony Algorithm [4], Genetic Algorithm [5], Tabu Search Algorithm [6] and etc. [7-10]. Some exact methods are parallelized on CPU in [11, 12] and GPU in [13]. On the other hand, parallel heuristic methods for KP are limited in the literature [14-16]. GPUs (Graphics Processing Units) are very powerful architecture for solving combinatorial optimization problem like KP. Compute Unified Device Architecture (CUDA) is parallel computation platform and programming model developed by NVIDIA. In this paper, a simulated annealing based algorithm is developed for solving 0/1 KP. This method is also parallelized on GPU using CUDA platform.

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. The 0/1 KP and its formulation are presented in Section 2, proposed method using Simulated Annealing Algorithm is described in Section 3. Section 4 gives details about parallelization on GPU. Computational experiments are presented in Section 5 and finally Section 6 deals with conclusion and future work.

## II. 0/1 KP

The 0/1 KP is one of the oldest combinatorial optimization problem. Several versions of KP such as 0/1 KP, multidimensional KP, multiple KP and KP sharing problems have been studied in the literature. For the 0/1 KP, given a set of $n$ objects, each object has a weight $w_i$ and profit $p_i$. The capacity of a knapsack is shown as $C$, the KP can be formulated mathematically as follows:

$$\max \sum_{i=1}^{n} p_i x_i \,,$$

$$s.t. \sum_{i=1}^{n} w_i x_i \leq C \qquad (1)$$

$$x_i \in \{0,1\}, i \in \{1, \dots, n\}$$

## III. SOLVING 0/1 KP USING SIMULATED ANNEALING ALGORITHM

Simulated Annealing (SA) algorithm has been developed by Kirkpatrick et al. [17] for solving economic activities in 1983. The aim of the SA is finding global maximum or minimum point of a function that has more than one local maximum or minimum point. The SA generally uses the Metropolis criterion and the Boltzmann distribution for


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

acceptance a new state. The acceptance probability function is shown in (2):

$$P(\Delta E) = \begin{cases} 1 & if\ \Delta E < 0, \\ exp(-\Delta E/T) & otherwise, \end{cases} \quad (2)$$

where T is temperature at the current solution. P is acceptation probability of neighborhood in annealing process. ΔE is the difference in energy between two neighbors solutions. The pseudocode of SA is as follows:

**Algorithm 1:** Pseudocode of Simulated Annealing Algorithm.

**Begin**
$p \leftarrow p_0$;
$e \leftarrow E(p)$;
$p_{best} \leftarrow p$;
$e_{best} \leftarrow e$;
**while** $T > T_{target}$ **and** stopping criterion is not met yet **do**
  $p_{new} \leftarrow random(p)$;
  $e_{new} \leftarrow E(p_{new})$;
  **if** $P(e, e_{new}, T) > random(0,1)$ **then**
  $p \leftarrow p_{new}$;
    $e \leftarrow e_{new}$;
  **end if**
  **if** $e < e_{best}$ **then**
  $p_{best} \leftarrow p_{new}$;
    $e_{best} \leftarrow e_{new}$;
  **end if**
$T \leftarrow T * cooling\_factor$;
**end while**
**End.**

In this paper, the proposed method using SA is used to determine the neighbor solution of a state. The steps of proposed method are given below:

1. Calculate the density ($d = p_i / w_i$) of each object in a given knapsack and sort by density in descending order.
2. Add objects to the knapsack by order until the capacity of the knapsack is not reached the maximum capacity.
3. Initialize SA Parameters ($T$, $T_{target}$, $\alpha$), *Total_Capacity*, *Total_Profit* and *Total_Profit_Best*.
4. Pick a random object *X* which is not in the knapsack.
5. Add object *X* into the knapsack. If *Total_Capacity* is less than the maximum capacity then continue with the next step, otherwise go to *Step 10*.
6. Pick a random object *Y* which is not in the knapsack. Remove object *X* from the knapsack and add object *Y* into the knapsack. If *Total_Capacity* is less than the maximum capacity then continue with the next step, otherwise go to *Step 9*.
7. Calculate the profit difference ($\Delta P = P_x$-$P_y$) between object *X* and object *Y*.
8. If the $\Delta P$ is less than zero or probability value calculated according to acceptance probability function is greater than a *random(0,1)*, select object *Y* instead of object *X* and add $P_y$ to *Total_Profit*. Otherwise go to the next step.
9. Select object *X* and add it into the knapsack. Add $P_x$ to *Total_Profit*. Go to *Step 15*.
10. Pick a random object *Z* in the knapsack.
11. Remove object *Z* from the knapsack. If *Total_Capacity* is less than the maximum capacity, then continue with the next step, otherwise go to *Step 14*.
12. Calculate the profit difference ($\Delta P = P_z$-$P_x$) between object *Z* and object *X*.
13. If the $\Delta P$ is less than zero or probability value calculated according to Metropolis criterion is greater than a *random(0,1)*, then select object *X* and add $P_x$ to *Total_Profit*. Otherwise go to the next step.
14. Select object *Z* instead of object *X* and add it into the knapsack again. Add $P_z$ to *Total_Profit*.
15. If *Total_Profit* is greater than *Total_Profit_Best* then assign it as best.
16. Perform cooling factor to *T* by $\alpha$.
17. Continue from *Step 4* until temperature $T_{target}$ is reached.
18. Obtain *Total_Profit_Best* and terminate the program.

## IV. Parallelization On GPU

Multi-start technique is efficient to solve any combinatorial problem with heuristic algorithms [18]. This technique is also used with the SA and provides better quality solutions than a single start technique [19, 20].

Proposed method that defined in Section 3 after Step 3, parallelized on GPU with multi-start technique. Every thread on GPU starts a different candidate object to achieve best profit for the knapsack with a given capacity. cuRAND is a pseudorandom number generator library on CUDA platform and 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 proposed method independently with given parameters for SA. At the end of the process, each thread in a block communicates each other using shared memory. The best profit value is found for each block in parallel using reduction method. The flowchart of the proposed method with parallel approach is shown in Fig. 1. Firstly, Step 1 and Step 2 run on a CPU. Then, Step 3 to Step 18 runs on a GPU by threads on blocks. Threads are organized in three different ways at GPU for this purpose. These organizations are given as follows:


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

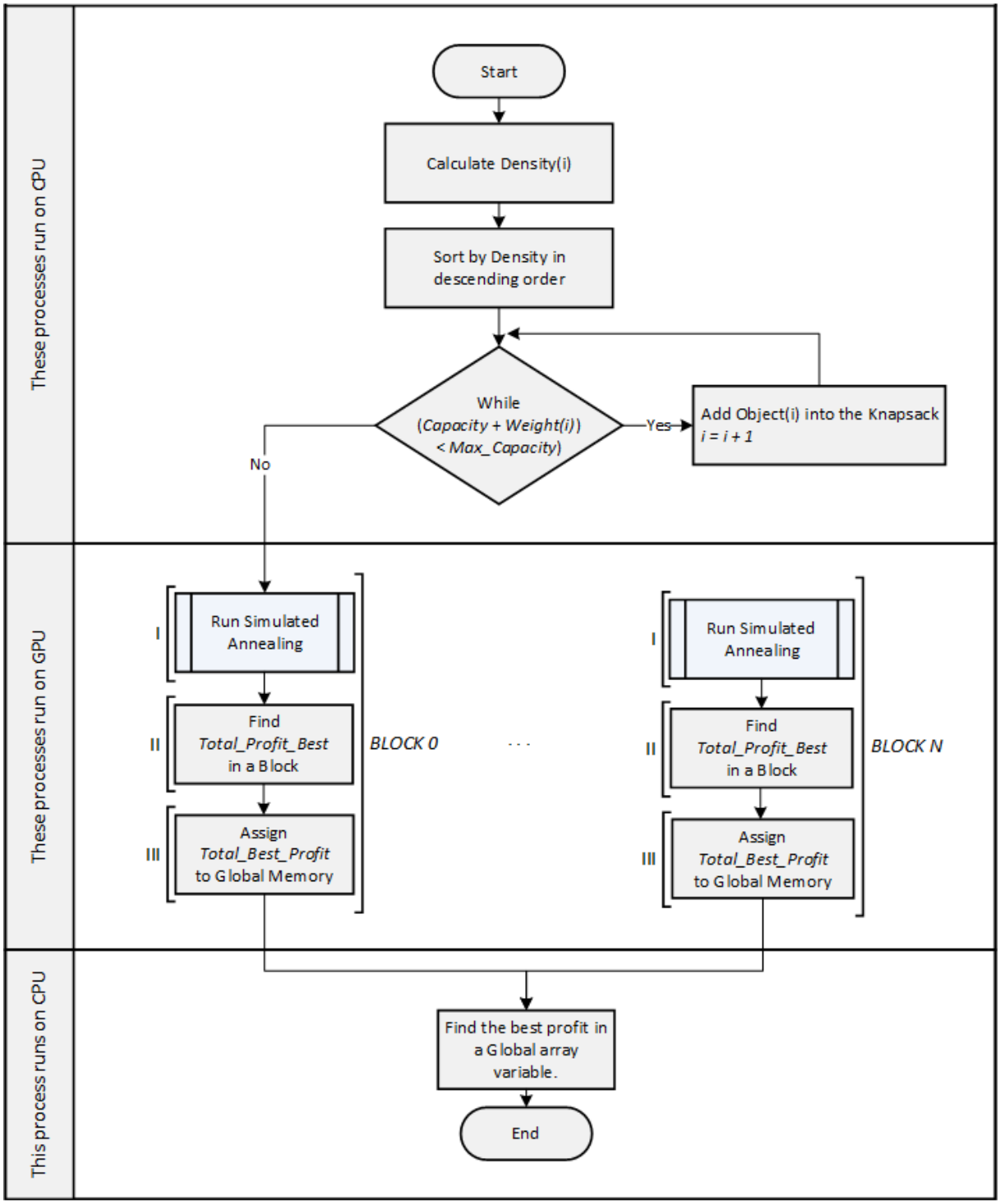


Figure 1. Flowchart of the proposed method in parallel.

I. All threads in a block are active.

II. The first half of the threads on a block is active.

III. The first thread on a block is active.

After the process of the GPU, the best profit value is obtained by CPU in a global array variable which contains best profit values of each block on the GPU.

There are 1024 threads per block on GPU in our work. All threads on a block are used. When the number of blocks is increased, runtime increases, too. All threads try to access the


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

global memory to get the objects in the knapsack. Since, the number of blocks used in this study is 1024, the number of threads is 1024x1024.

## V. Computational Experiments and Results

Computational tests have been carried out on several low and medium dimensional test problems. 10 standard low-dimensional test problems (KP1-KP10) are shown in Table 1. Six medium-dimensional test instances are randomly generated (KP11-KP16).

TABLE I. Standard Ten Low-Dimensional 0/1 KPs

| Problem | n | C | Optimum | Weights *w* & Profits *p* |
|---|---|---|---|---|
| KP1 | 10 | 269 | 295 | ***w*** = {95, 4, 60, 32, 23, 72, 80, 62, 65, 46}, ***p*** = {55, 10, 47, 5, 4, 50, 8, 61, 85, 87} |
| KP2 | 20 | 878 | 1024 | ***w*** = {92, 4, 43, 83, 84, 68, 92, 82, 6, 44, 32, 18, 56,83, 25, 96, 70, 48, 14, 58}, ***p*** = {44, 46,90, 72, 91, 40, 75, 35, 8, 54, 78, 40, 77, 15, 61, 17,75, 29, 75, 63} |
| KP3 | 4 | 20 | 35 | ***w*** = {6, 5, 9, 7}, ***p*** = {9, 11, 13, 15} |
| KP4 | 4 | 11 | 23 | ***w*** = {2, 4, 6, 7}, ***p*** = {6, 10, 12, 13} |
| KP5 | 15 | 375 | 481.0694 | ***w*** = {56.358531, 80.874050, 47.987304, 89.596240, 74.660482, 85.894345, 51.353496,1.498459, 36.445204, 16.589862, 44.569231, 0.466933, 37.788018, 57.118442, 60.716575}, ***p*** = {0.125126, 19.330424, 58.500931, 35.029145, 82.284005, 17.410810, 71.050142, 30.399487, 9.140294, 14.731285, 98.852504, 11.908322, 0.891140, 53.166295, 60.176397} |
| KP6 | 10 | 60 | 52 | ***w*** = {30, 25, 20, 18, 17, 11, 5, 2, 1, 1}, ***p*** = {20, 18, 17, 15, 15, 10, 5, 3, 1, 1} |
| KP7 | 7 | 50 | 107 | ***w*** = {31, 10, 20, 19, 4, 3, 6}, ***p*** = {70, 20,39, 37, 7, 5, 10} |
| KP8 | 23 | 10,000 | 9767 | ***w*** = {983, 982, 981, 980, 979, 978, 488, 976, 972, 486, 486, 972, 972, 485, 485, 969, 966, 483, 964, 963, 961, 958, 959}, ***p*** = {981, 980, 979, 978, 977, 976, 487, 974, 970, 485, 485, 970, 970, 484, 484, 976, 974, 482, 962, 961, 959, 958, 857} |
| KP9 | 5 | 80 | 130 | ***w*** = {15, 20, 17, 8, 31}, ***p*** = {33, 24, 36,37, 12} |
| KP10 | 20 | 879 | 1025 | ***w*** = {84, 83, 43, 4, 44, 6, 82, 92, 25, 83, 56, 18, 58, 14, 48, 70, 96, 32, 68, 92}, ***p*** = {91, 72, 90, 46, 55, 8, 35, 75, 61, 15, 77, 40, 63, 75, 29, 75, 17, 78, 40, 44} |

NVIDIA GeForce GTX Titan X graphic card (3072 cores, 1.0 GHz) has been used. The quality of solutions and parallel computational time are compared to sequential run obtained on CPU with Intel Xeon 2.4 GHz. All methods have been developed with C++ programming language.

The performance of the SA depends on the parameters belonging to the method. Parameter configurations are determined as $T = 1000$, $T_{target} = 1$, $\alpha = 0.95$. For each problem, results have been obtained by average of 10 independent runs on CPU. The results of proposed method on low-dimensional standard test problems are presented in Table 2. The best, the worst, the mean, the median and the standard deviation (SD) are listed on table for all problem instances. Proposed method obtained optimum solution for ten low-dimensional problems.

TABLE II. Results Of Proposed Method On KP1-KP10

| Problem | Best | Worst | Mean | Median | SD |
|---|---|---|---|---|---|
| KP1 | 295 | 295 | 295 | 295 | 0.00 |
| KP2 | 1024 | 1024 | 1024 | 1024 | 0.00 |
| KP3 | 35 | 35 | 35 | 35 | 0.00 |
| KP4 | 23 | 23 | 23 | 23 | 0.00 |
| KP5 | 481.0694 | 437.9345 | 472.4424 | 481.0694 | 17.25 |
| KP6 | 52 | 52 | 52 | 52 | 0.00 |
| KP7 | 107 | 81 | 100.5 | 105 | 7.53 |
| KP8 | 9767 | 9754 | 9755.4 | 9758 | 4.22 |
| KP9 | 130 | 130 | 130 | 130 | 0.00 |
| KP10 | 1025 | 1025 | 1025 | 1025 | 0.00 |

For medium-dimensional instances, *n* is set to 100, 200, 300, 500, 800 and 1000. For each *n*, weight (is between 5 and 20) and profit (is between 50 and 100) are generated randomly. The capacity of each knapsack is set to 1100, 1500, 1700, 2000, 5000 and 10000, respectively. Similarly, as low-dimensional instances, 10 independent runs are reported in Table 3. The best results are shown in bold.

TABLE III. Results Of Proposed Method On KP11-KP16

| Problem | n | Best | Worst | Mean | Median | SD |
|---|---|---|---|---|---|---|
| KP11 | 100 | **6673** | 6647 | 6666.6 | 6673 | 9.83 |
| KP12 | 200 | **11708** | 11660 | 11684.9 | 11684 | 18.39 |
| KP13 | 300 | **13658** | 13622 | 13652.4 | 13658 | 11.76 |
| KP14 | 500 | **18621** | 18594 | 18612.1 | 18613 | 6.86 |
| KP15 | 800 | **40111** | 40069 | 40106.8 | 40111 | 12.60 |
| KP16 | 1000 | **66249** | 66213 | 66241.0 | 66249 | 12.27 |

The proposed method in parallel runs on GPU only once. Table 4 represents the experimental results on both low-dimensional and medium-dimensional instances from run of the parallel approach (PA). Table 4 also shows the best and the mean of the sequential runs (S.Best & S.Mean) for comparing results. The best results are shown in bold in Table 4. For the


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

medium-dimensional instances, parallel approach has achieved better results. It is clear that parallel approach obtains good quality results than sequential one.

TABLE IV. RESULTS OF PROPOSED METHODS IN SEQUENTIAL AND PARALLEL ON KP1-KP16

| Problem | PA | S. Best | S. Mean |
|---|---|---|---|
| KP1 | **295** | **295** | **295** |
| KP2 | **1024** | **1024** | **1024** |
| KP3 | **35** | **35** | **35** |
| KP4 | **23** | **23** | **23** |
| KP5 | **481.0694** | **481.0694** | 472.4424 |
| KP6 | **52** | **52** | **52** |
| KP7 | **107** | **107** | 100.5 |
| KP8 | **9767** | **9767** | 9755.4 |
| KP9 | **130** | **130** | **130** |
| KP10 | **1025** | **1025** | **1025** |
| KP11 | **6711** | 6673 | 6666.6 |
| KP12 | **11738** | 11708 | 11684.9 |
| KP13 | **13668** | 13658 | 13652.4 |
| KP14 | **18644** | 18621 | 18612.1 |
| KP15 | **40121** | 40111 | 40106.8 |
| KP16 | **66270** | 66249 | 66241.0 |

For each problem, runtime results are given on Table 5 (Sequential Runtime-S.R., Parallel Runtime-P.R.) and the results for parallel approach have been obtained by average of 10 independent runs on GPU. According to time results of sequential method for KP1-KP10, runtimes are close to each other on instances and average time is 9.13 seconds. For KP11-KP16, results are close each other as low-dimensional instances and average time is 6.25 seconds.

TABLE V. TIMES IN SECONDS (S) OF PROPOSED METHODS IN SEQUENTIAL AND PARALLEL ON KP1-KP16

| Problem | S.R. (s) | P.R. (s) | Problem | S.R. (s) | P.R. (s) |
|---|---|---|---|---|---|
| KP1 | 9.03 | 0.87 | KP11 | 6.61 | 0.91 |
| KP2 | 8.81 | 0.84 | KP12 | 6.08 | 0.90 |
| KP3 | 9.13 | 0.83 | KP13 | 6.12 | 0.90 |
| KP4 | 12.40 | 0.79 | KP14 | 5.93 | 0.89 |
| KP5 | 6.84 | 0.82 | KP15 | 6.28 | 0.92 |
| KP6 | 7.06 | 0.80 | KP16 | 6.46 | 0.92 |
| KP7 | 6.74 | 0.80 | | | |
| KP8 | 11.36 | 0.82 | | | |
| KP9 | 11.09 | 0.77 | | | |
| KP10 | 8.82 | 0.87 | | | |
| *Average* | **9.13** | **0.82** | *Average* | **6.25** | **0.91** |

The average of low-dimensional instances is greater than the average medium-dimensional instances. This is due to the fact that likelihood of random numbers being equal is high for low-dimensional instances.

Speed-up criterion is used for comparing runtimes between sequential and parallel methods. Speed-ups are calculated according to a formula as below:

$$Speedup = \frac{time_{sequential}}{time_{parallel}} \quad (3)$$

Speed-up results were shown in Fig.2. Speed-up value is obtained up to 16x and results are varied for low-dimensional instances because of the likelihood of random numbers being equal is high. The average speed-up of low-dimensional instances is 11x and the average speed-up of medium-dimensional instances is 7x.

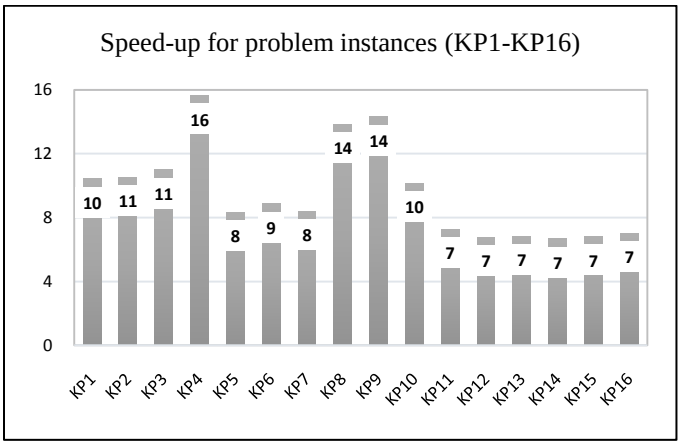


Figure 2. Speed-up results of problem instances.

## VI. CONCLUSIONS

In this paper, a parallel approach for solving the 0/1 KP using the SA algorithm is presented. Firstly, the proposed method runs in sequential and obtained the best known value in low-dimensional instances within a short time. Also the method runs in parallel on a GPU by using CUDA platform. The proposed method which runs in parallel is up to 16x faster than a single-core CPU. In terms of quality solutions, the proposed method in parallel is capable of delivering good quality results for both low-dimensional and medium-dimensional instances in a short time. In future, proposed method can be parallelized on GPU only and it can be applied to other types of KPs.

## ACKNOWLEDGMENT

The authors would like to thank Dr. Hakan Kutucu at Karabuk University for proof-reading the paper.

## REFERENCES

[1] S. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," Management Science, vol. 45, pp. 414-424, 1999.

[2] S. Sahni, "Approximate algorithms for the 0/1 knapsack problem," Journal of the ACM (JACM), vol. 22, pp. 115-124, 1975.


https://sites.google.com/site/ijcsis/
ISSN 1947-5500

[3] A. Liu, J. Wang, G. Han, S. Wang, and J. Wen, "Improved simulated annealing algorithm solving for 0/1 knapsack problem," in Sixth International Conference on Intelligent Systems Design and Applications, 2006, pp. 1159-1164.

[4] H. Shi, "Solution to 0/1 knapsack problem based on improved ant colony algorithm," in 2006 IEEE International Conference on Information Acquisition, 2006, pp. 1062-1066.

[5] G. R. Raidl, "An improved genetic algorithm for the multiconstrained 0-1 knapsack problem," in Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, 1998, pp. 207-211.

[6] S. Hanafi and A. Freville, "An efficient tabu search approach for the 0–1 multidimensional knapsack problem," European Journal of Operational Research, vol. 106, pp. 659-675, 1998.

[7] M. M. Akbar, E. G. Manning, G. C. Shoja, and S. Khan, "Heuristic solutions for the multiple-choice multi-dimension knapsack problem," in International Conference on Computational Science, 2001, pp. 659-668.

[8] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," European Journal of Operational Research, vol. 155, pp. 1-21, 2004.

[9] D. Zou, L. Gao, S. Li, and J. Wu, "Solving 0–1 knapsack problem by a novel global harmony search algorithm," Applied Soft Computing, vol. 11, pp. 1556-1564, 2011.

[10] X. Kong, L. Gao, H. Ouyang, and S. Li, "A simplified binary harmony search algorithm for large scale 0–1 knapsack problems," Expert Systems with Applications, vol. 42, pp. 5337-5355, 2015.

[11] D. El Baz and M. Elkihel, "Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0–1 knapsack problem," Journal of Parallel and Distributed Computing, vol. 65, pp. 74-84, 2005/01/01 2005.

[12] A. Goldman and D. Trystram, "An efficient parallel algorithm for solving the Knapsack problem on hypercubes," Journal of Parallel and Distributed Computing, vol. 64, pp. 1213-1222, 2004/11/01 2004.

[13] V. Boyer, D. El Baz, and M. Elkihel, "Solving knapsack problems on GPU," Computers & Operations Research, vol. 39, pp. 42-47, 1// 2012.

[14] P. Pospichal, J. Schwarz, and J. Jaros, "Parallel genetic algorithm solving 0/1 knapsack problem running on the gpu," in 16th International Conference on Soft Computing MENDEL, 2010, pp. 64-70.

[15] M. Hajarian, A. Shahbahrami, and F. Hoseini, "A parallel solution for the 0/1 knapsack problem using firefly algorithm," in 2016 1st Conference on Swarm Intelligence and Evolutionary Computation (CSIEC), 2016, pp. 25-30.

[16] H. Fingler, E. N. Cáceres, H. Mongelli, and S. W. Song, "A CUDA based solution to the multidimensional knapsack problem using the ant colony optimization," Procedia Computer Science, vol. 29, pp. 84-94, 2014.

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

[18] 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.

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

[20] 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.


https://sites.google.com/site/ijcsis/
ISSN 1947-5500