Int. j. inf. tecnol.
https://doi.org/10.1007/s41870-023-01328-1

ORIGINAL RESEARCH

A multicore-based algorithm for optimal multi-way number partitioning

Kamel M. K. Abdelsalam[1] · Soheir M. Khamis[1] · Hatem M. Bahig[1] · Hazem M. Bahig[1,2]

Received: 21 December 2022 / Accepted: 26 May 2023

Abstract The Multi-Way Number Partitioning (MWNP) problem is to divide a multiset of n numbers into k subsets in such a way that the largest subset sum is minimized. MWNP is an NP-hard combinatorial optimization problem which requires a high computational time. It has applications in many areas, such as processor scheduling and public key encryption. In this paper, we design a fast parallel algorithm that finds an exact solution for the MWNP problem based on the recursive principle of optimality. The practical studies on a multicore system and data set generated in the range $[1, 2^{48} - 1]$ show that the proposed parallel algorithm is able to reduce the running time of the corresponding sequential algorithm. The results of the parallel algorithm show that, using 16 cores, the algorithm achieves, on average, a speedup of 10 and a percentage 90% of improvement in running time over its sequential counterpart.

Keywords Number partitioning · Parallel algorithms · Multicore systems · Optimization, NP-hard

1 Introduction

In the recent years, different parallel systems have been developed and played a significant role in the computing field. One of the research fields that is affected by this technology is optimization problems. Examples of optimization problems are scheduling problem [1, 2], wireless sensor network [3], bioinformatics [4, 5], and steganography [6]. The main goals of using parallelism in optimization field are speeding up the computation and being able to solve problems of large sizes. Examples of using parallelism for different optimization problems in a variety of fields of research are addition chain [7, 8] and bioinformatics [9–12].

The multi-way number partitioning (MWNP) is one of the optimization problems that require parallel processing due to the high computational time to find the optimal solution. It aims to divide a multiset S of n numbers into k subsets such that the largest subset sum is minimized. Formally, MWNP problem is defined as in Table 1.

For example, given $S = \{3, 2, 4, 1, 3, 3, 6\}$ and $k = 3$. $P_1 = \langle\{6, 3\}, \{4, 3\}, \{3, 2, 1\}\rangle$ is a partition of S with $cost(P_1) = 6 + 3 = 9$, while $P_2 = \langle\{6, 2\}, \{4, 3\}, \{3, 3, 1\}\rangle$ is another partition of S with $cost(P_2) = 6 + 2 = 8$. The partition P_2 has a smaller cost than P_1, so P_2 is a better solution in this example. It is clear that there is no better solution than P_2 since its cost is the perfect cost $C_p = \left\lceil \frac{\sum_{x \in S} x}{k} \right\rceil = \left\lceil \frac{22}{3} \right\rceil = 8$. Hence, P_2 is optimal.

There are two main approaches to solving the MWNP problem. The first approach is based on solving the MWNP problem in a polynomial time but the output solution is not exact. Examples of approximation algorithms are the Longest Processing Time (LPT) algorithm [13] and the Karmarkar-Karp (KK) algorithm [14]. The second approach is based on finding an optimal solution for the MWNP problem. This approach requires a high computational time to find the solution. Examples of optimal solutions for MWNP

✉ Hazem M. Bahig
hbahig@sci.asu.edu.eg

[1] Computer Science Division, Mathematics Department, Faculty of Science, Ain Shams University, Cairo, Egypt

[2] Information and Computer Science Department, College of Computer Science and Engineering, Ha'il University, Ha'il, Kingdom of Saudi Arabia

Table 1 A formal definition of the MWNP problem

Input	A multiset, S, of n positive integers and a number of subsets, k, where $n, k \in \mathbb{N} - \{0\}$ and $n \geq k \geq 2$.
A Feasible Solution	$P = \langle S_1, S_2, \ldots, S_k \rangle$, where $S_i \subseteq S$: $i = 1, 2, \ldots, k, \bigcup_{i=1}^{k} S_i = S$, and $S_i \cap S_j = \phi$ for $i \neq j$. That is, P is a disjoint partition (simply, a partition) of S.
Cost	For a partition $P = \langle S_1, S_2, \ldots, S_k \rangle$ of S, the cost of P, denoted by $cost(P)$, is the largest subset sum in P, i.e. $cost(P) = \max_{i=1}^{k} \left\{ \sum_{x \in S_i} x \right\}$. (1) If $cost(P) = \left\lceil \frac{\sum_{x \in S} x}{k} \right\rceil$, then the cost is called perfect cost, denoted by C_p, and the partition P is called a perfect partition. Obviously, a perfect partition is optimal.
Goal	Minimization of the costs of all feasible solutions of MWNP.

are the Recursive Number Partitioning (RNP) algorithm [15] and the Improved RNP (IRNP) algorithm [16].

The goal of this paper is to reduce the execution time of one of the exact algorithms on a multicore system for two reasons: (1) The main drawback of the aforementioned algorithms is that the running time to get an optimal solution rapidly increases with a small increase in the number of elements, n, in the multiset S. (2) There is no parallel algorithm for optimally solving the MWNP problem.

In this paper, we focus on parallelizing the IRNP algorithm by distributing the search space among many processors. The experimental results show the effectiveness of parallelization and the improvement of the running time by percentage 90%, using 16 threads, over the IRNP algorithm.

The rest of the paper is organized as follows. In Sect. 2, a quick review of the previous works on MWNP problem. In Sect. 3, the sequential algorithm IRNP is described. Section 4 is dedicated to describe the main idea and pseudocode of the proposed parallel algorithm. In Sect. 5, the experimental results of measuring the performance of the proposed parallel algorithm are presented. Finally, Sect. 6 concludes the work done in this paper and suggests some future works.

2 Related work

A large number of algorithms are proposed to solve MWNP problem. These algorithms categorized into two groups to solve the MWNP problem. The first group includes approximation algorithms that solve MWNP in a polynomial time such as the Longest Processing Time (LPT) algorithm [13], the Karmarkar-Karp (KK) algorithm [14], and the Cached Iterative Weakening (CIW) algorithm [17]. The LPT algorithm is a greedy algorithm for solving the MWNP problem that runs in time $O(n \log n + n)$. Karmarkar and Karp have defined a set of differencing operations [14] for MWNP and introduced their set differencing algorithm which is based on these operations. Both algorithms, LPT and KK, find an optimal solution when $n \leq k + 2$, where $n \geq k \geq 2$. Otherwise, their output is used as an upper bound, ub, in the exact algorithms. The KK algorithm has the same order of time complexity as the LPT algorithm however, in general, the KK algorithm obtains a better solution than the LPT algorithm [18–20].

The CIW algorithm is a parametric algorithm that assumes the existence of a perfect partition and hence sets the bounds. Then, it iteratively weakens these bounds until it finds a feasible solution which is guaranteed to be near-optimal. The CIW algorithm uses a modified version of the Extended Schroeppel-Shamir (ESS) algorithm [16] that requires to be given the number of subsets to be generated, as a parameter, in advance. Choosing this parameter is a problem that rises when applying the CIW algorithm [19, 20] to solve the MWNP problem. However, the CIW algorithm outperforms all heuristic algorithms for MWNP problem.

The second group includes exact (optimal) algorithms such as the Horowitz-Sahni (HS) algorithm [21] and the Schroeppel-Shamir (SS) algorithm [22] for the 2-way number partitioning which is a special case of the MWNP problem when $k = 2$. For the general case of multi-way partitioning, there are the Recursive Number Partitioning (RNP) algorithm [15], the Improved RNP (IRNP) algorithm [16], the Moffitt algorithm [23], and the Sequential Number Partitioning (SNP) algorithm [24].

Both the HS and SS algorithms run in the same time complexity $O(n \cdot 2^{\frac{n}{2}})$ [19]. However, The SS algorithm runs 10 times faster than the HS algorithm for $n > 55$ [19, 20]. Also, the HS algorithm uses $O(2^{\frac{n}{2}})$ of space, while the SS algorithm has a lower space complexity $O(2^{\frac{n}{4}})$ [19, 20, 25].

Extended versions of the HS and SS algorithms, namely the Extended HS (EHS) and Extended SS (ESS) algorithms

[16], are used in some of the algorithms for solving the MWNP problem as subroutines to generate subsets whose sums lie in a given range. Also, the Inclusion–Exclusion (IE) algorithm [15] is used for the same purpose. The IE algorithm is a binary search tree algorithm that runs in $O(2^n)$ and uses $O(n)$ of space. Clearly, the ESS algorithm is faster than both the HS and IE algorithms.

The RNP algorithm uses the IE algorithm to generate all subsets, one by one, in a specified range so that the complement of each generated subset can be recursively sub-partitioned into $k-1$ subsets if k is odd. Otherwise, the RNP algorithm uses an optimal 2-way partitioning algorithm called the Complete Karmarkar-Karp (CKK) algorithm [25] to partition S into two subsets, S^{k_1} and S^{k_2}, which are recursively sub-partitioned into k_1 and k_2, respectively, where $k_1 = k_2 = \frac{k}{2}$ [15].

The IRNP algorithm, the improved version of the RNP algorithm, uses the ESS algorithm, instead of the IE algorithm, to generate all subsets, one by one, in a specified range such that each generated subset is recursively sub-partitioned into $k_1 = \lfloor \frac{k}{2} \rfloor$ subsets then the complement of each generated subset is sub-partitioned into $k - k_1$ subsets. Both RNP and IRNP rely on the recursive principle of optimality [16, 19, 20].

The Moffitt algorithm uses the IE algorithm as the RNP algorithm but relies on the principle of weakest-link optimality [19, 20, 23]. The IRNP algorithm outperforms the Moffitt algorithm for small k's ($k \leq 5$) while the Moffitt algorithm outperforms the IRNP algorithm as when increasing k [19, 20].

The SNP algorithm has the same recursive decomposition concept as the Moffitt algorithm but uses the ESS algorithm instead of the IE algorithm to generate all subsets with sums in a given range. The SNP algorithm outperforms the Moffitt algorithm for $k \leq 6$ while the Moffitt algorithm outperfroms the SNP algorithm for $k \geq 7$ [19, 20]. All the previously mentioned multi-way partitioning algorithms use an initial upper and lower bounds and then tighten these bounds until an optimal solution is found.

3 Improved recursive number partitioning

In this section, we describe the main concepts of the IRNP algorithm and explain the steps of the algorithm.

3.1 Overview of IRNP

The IRNP algorithm is a branch-and-bound algorithm that starts with a solution from an approximation algorithm. Then, it carries on to find a better solution. The IRNP algorithm terminates if it finds a perfect partition or when the search space is exhausted which verifies that the last solution found is optimal [16]. The IRNP algorithm depends on the recursive principle of optimality [16] which can be simply expressed as follows. To optimally partition S into k subsets, first partition S into two subsets S^{k_1} and S^{k_2}, where $k_1 = \lfloor \frac{k}{2} \rfloor$ and $k_2 = k - k_1$. Then, recursively optimally partition S^{k_1} into k_1 subsets and S^{k_2} into k_2 subsets. The subset with smaller cardinaliy of S^{k_1} and S^{k_2} is chosen to be recursively partitioned first, since it will quickly fail or achieve a better solution [19, 20].

The IRNP algorithm can be viewed as a hybrid algorithm because it involves the following subroutines:

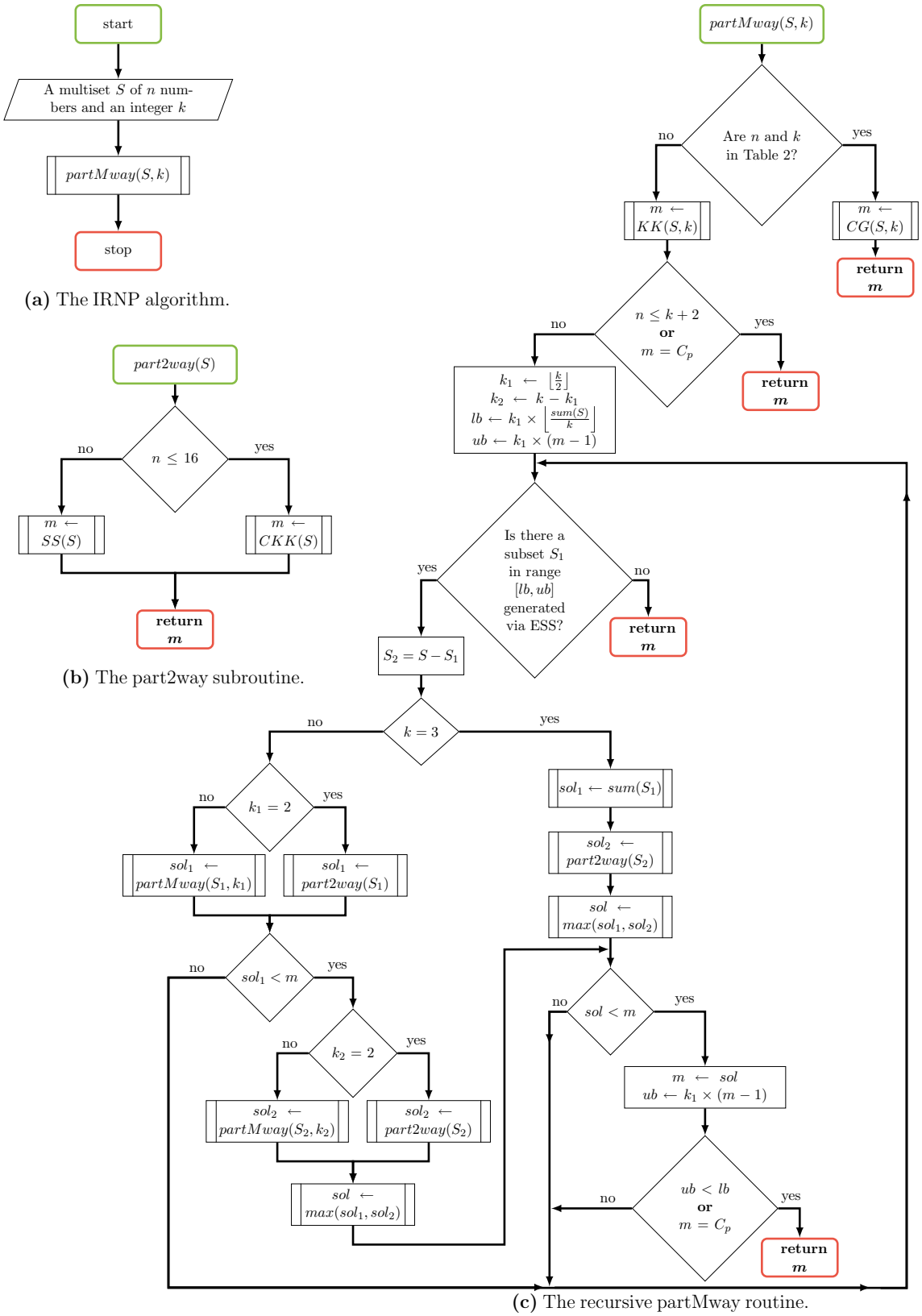
Complete Greedy Algorithm (CG): The CG algorithm [25] is an optimal branch-and-bound algorithm based on a k-ary tree. The leftmost branch in the k-ary tree represents the solution gained from the LPT algorithm [13] for the same problem. For $k = 2$, the CG algorithm performs a depth-first search on a binary tree. That is, it runs in time $O(2^n)$ and uses $O(n)$ of space. For MWNP, the CG algorithm is used for finding an optimal solution for the instances from Table 2 [16, 19, 20].

Karmarkar-Karp's Algorithm (KK): The KK algorithm [14] is a polynomial time approximation algorithm that is used for calculating the upper bounds applied throughout the algorithm.

Complete Karmarkar-Karp's Algorithm (CKK): The CKK algorithm [25] is an optimal branch-and-bound algorithm that extends the KK algorithm. It constructs k-ary tree in which the leftmost branch is the solution of the KK algorithm. For 2-way partitioning, the CKK algorithm is used for $5 \leq n \leq 16$ [19, 20], where it performs a depth-first search on a binary tree. That is, it runs in time $O(2^n)$ and uses $O(n)$ of space.

Table 2 The values of k and their corresponding values of n for which the CG algorithm is used

$k =$	3	4	5	6	7	8	9	10
$n \leq$	12	14	16	19	21	25	27	31



(a) The IRNP algorithm.

(b) The part2way subroutine.

(c) The recursive partMway routine.

Fig. 1 The flowcharts of the IRNP algorithm

Schroeppel-Shamir's Algorithm (SS): The SS algorithm [22] is an optimal 2-way partitioning algorithm that is used for $n \geq 17$ [19, 20].

Part2way subroutine: The part2way subroutine (Fig. 1(b)) combines the CKK and SS subroutines. It is called in IRNP whenever a subset is required to be partitioned into 2 subsets (2-way partitioning) and it returns the max of the two subset sums.

Extended Schroeppel-Shamir's Algorithm (ESS): The ESS algorithm [16] extends SS to function as a generator of subset sums that exist in a given range $[lb, ub]$ for the IRNP algorithm.

3.2 IRNP steps description

Figure 1 shows flowcharts of the IRNP algorithm. The figure includes the flowchart (Fig. 1a) for the IRNP algorithm showing its input and main recursive routine, while the flowchart (Fig. 1b) is for the part2way subroutine used for 2-way partitioning, and finally the flowchart (Fig. 1c) for the main recursive routine in the IRNP algorithm whose steps are described in the following.

The CG algorithm is applied if n and k are related together using the values in Table 2. In this case, the algorithm returns the output solution of the CG algorithm, which is optimal, and terminates. Otherwise, the algorithm computes an approximation solution using the KK algorithm. Based on the output of the KK algorithm, there are two cases:

1. The algorithm returns an optimal solution and terminates if either $n \leq k+2$ or the KK algorithm returns a perfect partition.
2. The algorithm uses the output of the KK algorithm as an initial value of the best partition cost found so far, denoted by m, and calculates a lower bound, lb, and an upper bound, ub, using the following formulae [16, 19, 20]:

$$lb = k_1 \times \left\lfloor \frac{\sum_{x \in S} x}{k} \right\rfloor, \quad ub = k_1 \times (m-1), \quad \text{where} \quad k_1 = \left\lfloor \frac{k}{2} \right\rfloor. \tag{2}$$

The algorithm divides S into two subsets S_1, which is generated by the ESS algorithm with sum in $[lb, ub]$, and $S_2 = S - S_1$. Then, it recursively sub-partitions S_1 and S_2 into k_1 and k_2 subsets, respectively, where $k_1 = \lfloor \frac{k}{2} \rfloor$ and $k_2 = k - k_1$. The algorithm, then, has the following two cases.

The first case is when $k = 3$. The algorithm finds the maximum value, denoted by sol, between the sum of S_1, since $k_1 = 1$, and the output of the 2-way partitioning algorithm (Fig. 1b) of S_2, since $k_2 = 2$. If $sol < m$, then the algorithm updates m and ub according to (Eq. 2). If $sol \geq m$, then the ESS algorithm generates another S_1 and the previous steps are repeated.

The second case is to apply the 2-way partitioning algorithm (Fig. 1b) to S_1 if $k_1 = 2$. Otherwise, the algorithm calls itself on S_1 and k_1 and the output of either cases is saved in sol_1. If $sol_1 < m$, the same is applied to S_2 and k_2 and the output is saved in sol_2. Then, the algorithm finds the maximum value, denoted by sol, between sol_1 and sol_2 and updates m and ub if $sol < m$. Otherwise, the ESS algorithm generates another S_1 and the previous steps are repeated. The IRNP algorithm continues in this manner until $lb > ub$, m is the perfect cost, or there is no more subsets in the search space [19, 20].

4 Parallel IRNP

In this section, we elaborate on how to parallelize the IRNP algorithm on a parallel shared memory model. In the first subsection, the main idea of parallelizing IRNP is described. In the second subsection, the pseudocode of the proposed PIRNP, parallel IRNP, algorithm is presented.

4.1 The idea of parallelizing IRNP

The idea of parallelizing the IRNP algorithm is based on how the proposed algorithm works on many subsets simultaneously. So, instead of generating one subset S_1 using the ESS algorithm, the proposed parallel algorithm generates a number $\alpha \in \mathbb{N} - \{0\}$ of subsets S_1. We discuss setting α experimentally in Sect. 5.2. After that, these α subsets are distributed among the available threads dynamically.

According to the mechanism of dynamic threads, the parallel system assigns one subset S_1 from the α subsets to each thread. Each thread works independently on its subset as in the sequential way. The result of each working thread is one of the following:

1. it finds and returns a perfect partition cost and the search process is terminated.
2. it returns the best partition cost found since there is no more subsets in the range of search (the search space is exhausted).
3. form the α subsets, a new subset S_1 is assigned to the available thread.

If all the α subsets are assigned to the available threads without finding an optimal solution, then another α subsets are generated and the same process is repeated.

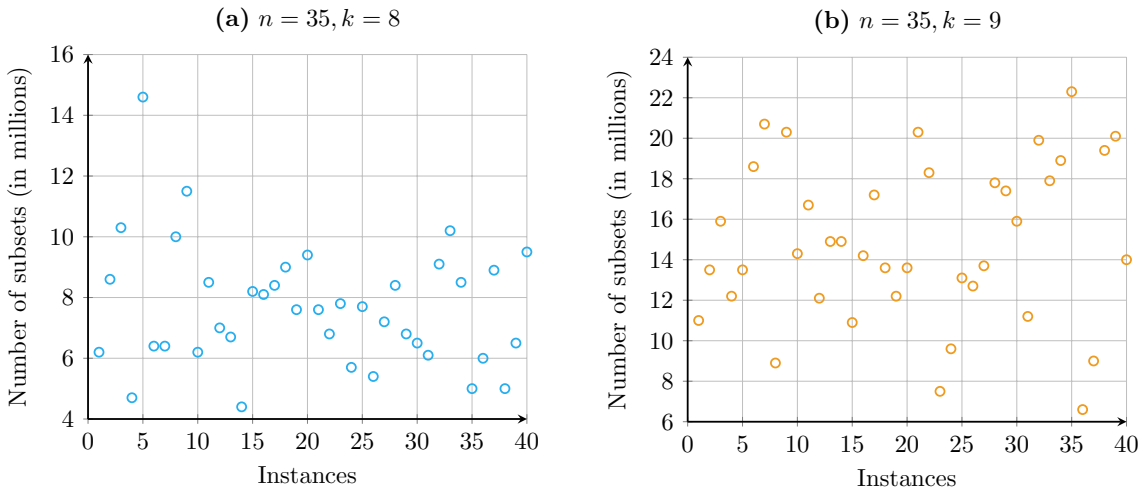


Fig. 2 The number of subsets generated when the IRNP algorithm calls the ESS algorithm for the first time

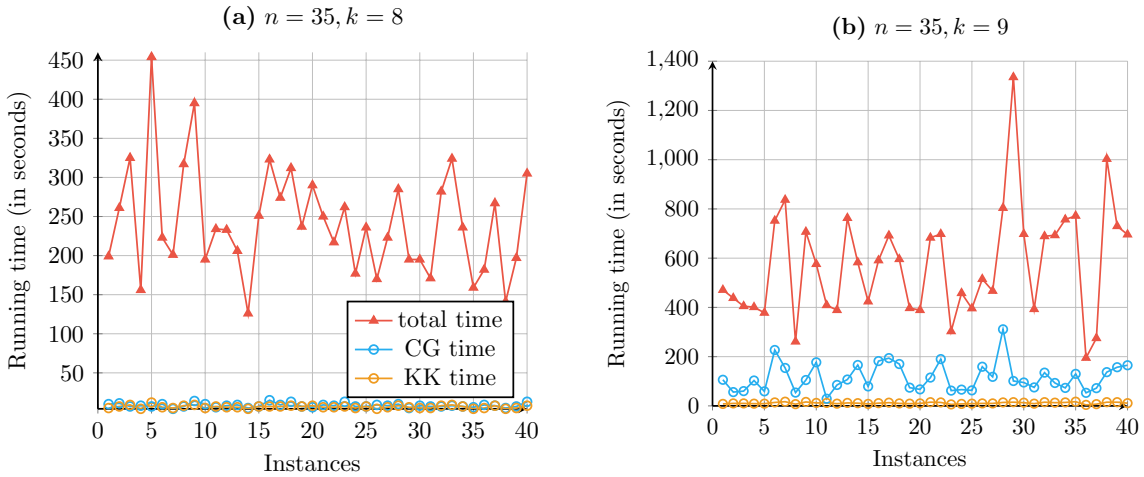


Fig. 3 The total time of the IRNP algorithm compared to the time of the two subroutines CG and KK

The algorithm uses "*found*" as a global shared variable that is initialized to be false. If a thread finds a perfect partition or the search space has no subsets anymore, then the thread updates the shared variable, *found*, to true. Also, before any thread works on its assigned subset S_1, it first tests if the shared variable found is still false.

The reasons for using the parallelism on the set of subsets are:

1. The total number of subsets obtained from the first call of the ESS algorithm is large. Figure 2 shows the number of subsets generated at the first call of the ESS algorithm when $n = 35$ and $k = 8$ and 9 for 40 instances.
2. The running time for calling subroutines such as the CG and KK algorithms is small compared to the total time as illustrated in Fig. 3.

At the end, the ESS algorithm may generate a number $\bar{\alpha} < \alpha$ subsets which are also assigned dynamically to the available threads.

4.2 PIRNP pseudocode

The pseudocode for the PIRNP algorithm is given in Algorithm 1. A brief description of the algorithm is explained in the following. Line 2 represents the calling of the CG algorithm to find an optimal solution when n and k verify the relation of Table 2 and the algorithm is terminated. Otherwise, Line 3 represents the calling of the KK algorithm to find an approximation or an optimal solution.

Algorithm 1 PIRNP

Input: A multiset, S, of n positive integers and a number of subsets, k.
Output: The cost of an optimal partition $P =< S_1, S_2, \ldots, S_k >$ of S.
(The variables m, lb, ub, C_p, and $found$ are global and shared throughout the program).

procedure PIRNP(S, k)
  if n and k are in Table 2 **then** **return** $CG(S, k)$ ▷ The algorithm terminates.
  $m \leftarrow KK(S, k)$
  if $n \leq k + 2$ **then** **return** m ▷ The algorithm terminates.
  $C_p \leftarrow \left\lceil \frac{sum(S)}{k} \right\rceil$
  if $m = C_p$ **then** **return** m ▷ The algorithm terminates.
  $k_1 \leftarrow \lfloor \frac{k}{2} \rfloor$
  $k_2 \leftarrow k - k_1$
  $lb = k_1 \times \left\lfloor \frac{sum(S)}{k} \right\rfloor$
  $ub \leftarrow k_1 \times (m - 1)$
  $found \leftarrow false$
  while found = **false do**
    ESS generates a number α of subsets $\{S_1^0, S_1^1, \ldots, S_1^{\alpha-1}\}$ with sums in range $[lb, ub]$
    for $i \leftarrow 0$ to $\alpha - 1$ **parallel do**
      if found = **false then**
        $S_1 \leftarrow S_1^i$ ▷ S_1^i is the i^{th} subset of the α subsets generated.
        $S_2 \leftarrow S - S_1$ ▷ S_2 is the complement of S_1.
        if k = 3 **then**
          $sol_1 \leftarrow sum(S_1)$
          $sol_2 \leftarrow part2way(S_2)$ ▷ Figure 1 (b)
          $sol \leftarrow max(sol_1, sol_2)$
          if $sol < m$ **then**
            $m \leftarrow sol$
            $ub \leftarrow k_1 \times (m - 1)$
            if $ub < lb$ **or** $m = C_p$ **then** $found \leftarrow$ **true**
          end if
        else
          if $k_1 = 2$ **then**
            $sol_1 \leftarrow part2way(S_1)$ ▷ Figure 1 (b)
          else
            $sol_1 \leftarrow partMway(S_1, k_1)$ ▷ Figure 1 (c)
          end if
          if $sol_1 < m$ **then**
            if $k_2 = 2$ **then**
              $sol_2 \leftarrow part2way(S_2)$ ▷ Figure 1 (b)
            else
              $sol_2 \leftarrow partMway(S_2, k_2)$ ▷ Figure 1 (c)
            end if
            $sol \leftarrow max(sol_1, sol_2)$
            if $sol < m$ **then**
              $m \leftarrow sol$
              $ub \leftarrow k_1 \times (m - 1)$
              if $ub < lb$ **or** $m = C_p$ **then** $found \leftarrow$ **true**
            end if
          end if
        end if
      end if
    end for
  end while
  return m ▷ The algorithm terminates.
end procedure

Table 3 The average time in seconds of PIRNP, for $n = 40$, in the four possible cases of the value of α and scheduling type

		α	Static Scheduling				Dynamic Scheduling			
			2	4	8	16	2	4	8	16
$k = 6$	Fixed Alpha	40	51.96	30	18.86	16.31	49.55	27.14	17.58	13.29
		80	50.62	28.65	17.85	14.5	49.05	26.48	15.62	11.51
		120	49.39	26.69	16.03	11.31	50.24	28.26	17.35	14.1
		160	50.37	28	17.02	13.78	49.02	26.25	15.19	11.03
		200	50.33	28.13	17.04	13.53	48.87	26.14	15.06	10.74
		240	66.47	39.15	22.14	13.42	61.83	32.86	17.79	10.67
		280	64.54	38.73	22.36	13.25	61.78	32.63	17.74	10.61
		320	65.99	38.51	22.11	13.2	61.92	32.74	17.7	10.51
	Variable Alpha	$10 \times nTh$	52.92	30.44	18.12	14.1	50.5	27.14	15.62	11.03
		$20 \times nTh$	52.34	29.13	17.26	13.42	49.55	26.48	15.19	10.44
		$30 \times nTh$	51.76	28.75	17.18	13.48	49.32	26.69	15	10.35
		$40 \times nTh$	51.02	28.21	16.75	12.98	49.05	26.25	14.92	10.29
		$50 \times nTh$	51.11	28.32	16.69	13.26	**49.16**	**26.14**	**14.86**	**10.32**
		$60 \times nTh$	68.87	39.26	22.12	13.02	63.09	33.04	17.48	10.19
		$70 \times nTh$	68.79	38.87	22.2	13.17	62.62	32.97	17.43	10.23
		$80 \times nTh$	67.47	39.01	22.19	13.05	63.22	32.75	17.74	10.13
$k = 7$	Fixed Alpha	40	194.29	128.6	94.31	84.98	167.75	102.57	77.16	69.07
		80	192.51	116.64	83.42	74.75	166.07	99.51	73.8	60.69
		120	202.88	121.3	86.52	72.5	166.13	99.38	71.53	59.93
		160	196.82	114.12	79.35	71.48	165.55	98.19	70.63	58.66
		200	199.14	113.67	82.54	73.14	165.17	96.75	68.87	57.04
		240	239.98	152.46	103.11	75.31	222.04	128.02	84.77	59.32
		280	251.42	149.19	104.15	76.8	220.23	127.52	84.62	58.78
		320	247.23	152.06	103.25	72.78	221.06	130.77	84.72	59.47
	Variable Alpha	$10 \times nTh$	199.55	128.12	81.68	71.02	173.45	103.73	74.5	59.72
		$20 \times nTh$	195.35	115.19	80.21	69.65	169.69	100.64	71.34	55.75
		$30 \times nTh$	194.75	120.98	78.08	66.59	169.19	100.58	69.78	55.55
		$40 \times nTh$	187.16	115.08	80.59	71.3	168.03	99.27	69.48	55.1
		$50 \times nTh$	179.09	114.71	78.88	69.49	**168.22**	**97.84**	**69.12**	**54.87**
		$60 \times nTh$	232.49	145.32	92.54	66.06	213.13	127.44	84.56	58.43
		$70 \times nTh$	234.84	144.18	96.08	67.17	215.62	125.7	83.27	59.06
		$80 \times nTh$	233.19	149.22	99.17	71.05	218.76	127.67	83.71	58.67

The optimal solution is obtained either when $n \leq k + 2$ [14, 18–20] or when the output of the KK algorithm matches the perfect solution as shown in lines 4-6. If the KK solution is not optimal, the algorithm determines the sizes k_1 and k_2 of the two subsets S_1 and S_2 of S, and the range of search $[lb, ub]$ for the subsets S_1's as given in lines 7-10. Lines 11-49 represent the process of searching for the perfect solution or the best partition cost that can be obtained for n and k. The searching process starts with generating α subsets by calling the ESS algorithm as in line 13. Then, the α subsets are distributed among the available threads to try finding an optimal solution as given in lines 14-48.

5 Experimental results and discussion

In this section, we study the performance of the proposed algorithm, PIRNP, compared to the sequential algorithm, IRNP. In the first subsection, the hardware and software configurations used in the experiments are described. In the second subsection, we discuss experimentally setting α and choosing the scheduling type. In the third subsection, the results of comparing the two algorithms, IRNP and PIRNP, in terms of the running time are discussed. Finally in the fourth subsection, the scalability of the proposed algorithm, PIRNP, is presented.

5.1 Platform and input dataset configuration

The results of implementing the IRNP and PIRNP algorithms on a machine that has 2 Intel Xeon E5645 processors. Each processor is running at 2.40GHz and able to execute 12 threads concurrently using 6 hyper-thread cores. The machine has 24GB of RAM and is running Linux Mint 20.2 (Uma) - Cinnamon (64-bit).

The algorithms are implemented in the C++ programming language and compiled using the GCC−11.1 Compiler. We also used the OpenMP library in the parallel algorithm to utilize the shared memory of the machine.

We used the same data set used in [19, 20]. The data set is generated randomly from the range $[1, 2^{48} - 1]$ as uniform distribution.

5.2 Best value of α and scheduling type

In this subsection, we estimate two factors that have an effect on the performance of the implementation of the parallel algorithm: the best value of α and the best scheduling type, for the parallel for-loop in Line 14 in Algorithm 1, used for handling the repeatedly generated α subsets S_1's using the available threads.

For the best value of α, we have the following choices:

1. a fixed value of α; i.e., α is a fixed number regardless any factor such as the number of threads. Experimentally, we set $\alpha = 40, 80, \ldots, 320$.
2. a variable value of α; i.e., α is dependent on certain factors. In this study, we set α depending on the number of threads.

$$\alpha = c \times nTh, \tag{3}$$

where nTh is the number of threads used, and $c = 10, 20, \ldots, 80$.

To decide the best scheduling type, we run the parallel algorithm using mechanisms, namely static and dynamic. The first one is that the generated α subsets are assigned to the available threads statically. This means that each thread will handle approximately $\left(\frac{\alpha}{nTh}\right)$ subsets.

The second mechanism is that the generated α subsets are handled dynamically. This means that, if a thread finishes its task, it is automatically assigned a new task if exists.

Based on the two cases of α and the two cases of the mechanism of the parallel for-loop, we have four possible cases. The results of these cases are shown in Table 3 for $n = 40$ and $k = 6$ and 7. As shown in the table, the best estimation of the two factors is the combination of variable value of α and the dynamic mechanism for the parallel for-loop.

Table 4 The average time, in seconds, for the IRNP and PIRNP algorithms

		IRNP	PIRNP							
			nTh							
n	k		2		4		8		16	
30	6	1.08	0.52	(51.9%)	0.31	(71.3%)	0.17	(84.3%)	0.11	(89.8%)
	7	3.29	1.7	(48.3%)	0.97	(70.5%)	0.56	(83.0%)	0.36	(89.1%)
	8	23.9	14.9	(37.7%)	7.7	(67.8%)	3.9	(83.7%)	2.1	(91.2%)
	9	40.4	24.9	(38.4%)	13.8	(65.8%)	8.2	(79.7%)	5	(87.6%)
35	6	10.08	4.95	(50.9%)	2.64	(73.8%)	1.43	(85.8%)	0.86	(91.5%)
	7	30.3	16.2	(46.5%)	9.1	(70.0%)	5.4	(82.2%)	3.7	(87.8%)
	8	217	122	(43.8%)	62	(71.4%)	31	(85.7%)	16	(92.6%)
	9	505	302	(40.2%)	174	(65.5%)	100	(80.2%)	69	(86.3%)
40	6	137	52	(62.0%)	27	(80.3%)	16	(88.3%)	11	(92.0%)
	7	508	218	(57.1%)	130	(74.4%)	72	(85.8%)	55	(89.2%)
	8	3165	1113	(64.8%)	567	(82.1%)	306	(90.3%)	192	(93.9%)
	9	8075	4218	(47.8%)	2983	(63.1%)	1983	(75.4%)	1370	(83.0%)
45	6	1414	634	(55.2%)	337	(76.2%)	185	(86.9%)	108	(92.4%)
	7	6359	3519	(44.7%)	1788	(71.9%)	1158	(81.8%)	979	(84.6%)
	8	36527	21786	(40.4%)	11010	(69.9%)	5839	(84.0%)	3067	(91.6%)
	9	142962	104872	(26.6%)	61106	(57.3%)	42333	(70.4%)	25037	(82.5%)
50	6	20398	10968	(46.2%)	5923	(71.0%)	3172	(84.4%)	1704	(91.6%)
	7	38423	23718	(38.3%)	17550	(54.3%)	13007	(66.1%)	6807	(82.3%)
	8	594304	308405	(48.1%)	175873	(70.4%)	99742	(83.2%)	50601	(91.5%)
	9	–	–	–	–	–	–	–	–	–

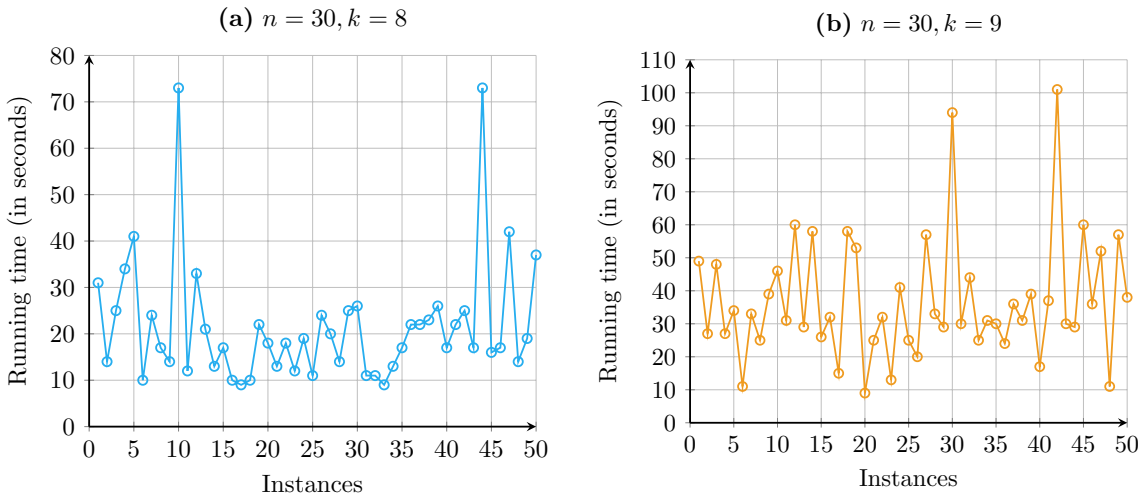


Fig. 4 The running time of IRNP is affected by the elements of the multiset

5.3 Running time comparison

The results of implementing the two algorithms, sequential and parallel, are shown in Table 4. For each fixed n and k, the running time of the IRNP and PIRNP algorithms is the average of 50 instances except for the following cases due to the large execution time: (1) $n = 45$ and $k = 8$ and 9, only 10 instances were used to compute the average, and (2) $n = 50$ and $k = 6, 7, 8$, only 5 instances were used to compute the average.

From Table 4, the following observations are made:

1. The running time of the PIRNP algorithm is less than that of the IRNP algorithm using any number of threads $nTh \geq 2$ for every n and k.
2. The percentage of improvement, poi_{nTh}, of the PIRNP algorithm using $nTh \geq 2$ threads is calculated using the formula in (Eq. 4) and is shown between brackets to the right of the average time of the PIRNP algorithm for every nTh.

$$poi_{nTh} = \left(1 - \frac{T_{nTh}}{T_s}\right) \times 100\%, \tag{4}$$

where T_{nTh} is the execution time of a parallel algorithm, PIRNP, using nTh threads and T_s is the execution time of the corresponding sequential algorithm, IRNP.

For example, when $n = 40$ and $k = 7$ the percentage of improvement for the PIRNP algorithm using 2 threads compared to the IRNP algorithm is 57.1%.

3. The performance of the PIRNP algorithm increases when the number of threads increases. This means that the running time of the PIRNP algorithm decreases when the number of threads increases.
4. Increasing the value of n with fixed k or increasing the value of k with fixed n leads to increasing the running time of the algorithms.

Table 5 The speedup of the PIRNP algorithm over the IRNP algorithm

		nTh			
n	k	2	4	8	16
30	6	2.08	3.48	6.35	9.82
	7	1.94	3.39	5.88	9.14
	8	1.60	3.10	6.13	11.38
	9	1.62	2.93	4.93	8.08
35	6	2.04	3.82	7.05	11.72
	7	1.87	3.33	5.61	8.19
	8	1.78	3.50	7.00	13.56
	9	1.67	2.90	5.05	7.32
40	6	2.63	5.07	8.56	12.45
	7	2.33	3.91	7.06	9.24
	8	2.84	5.58	10.34	16.48
	9	1.91	2.71	4.07	5.89
45	6	2.23	4.2	7.64	13.09
	7	1.81	3.56	5.49	6.50
	8	1.68	3.32	6.26	11.91
	9	1.36	2.34	3.38	5.71
50	6	1.86	3.44	6.43	11.97
	7	1.62	2.19	2.95	5.64
	8	1.93	3.38	5.96	11.74
	9	–	–	–	–

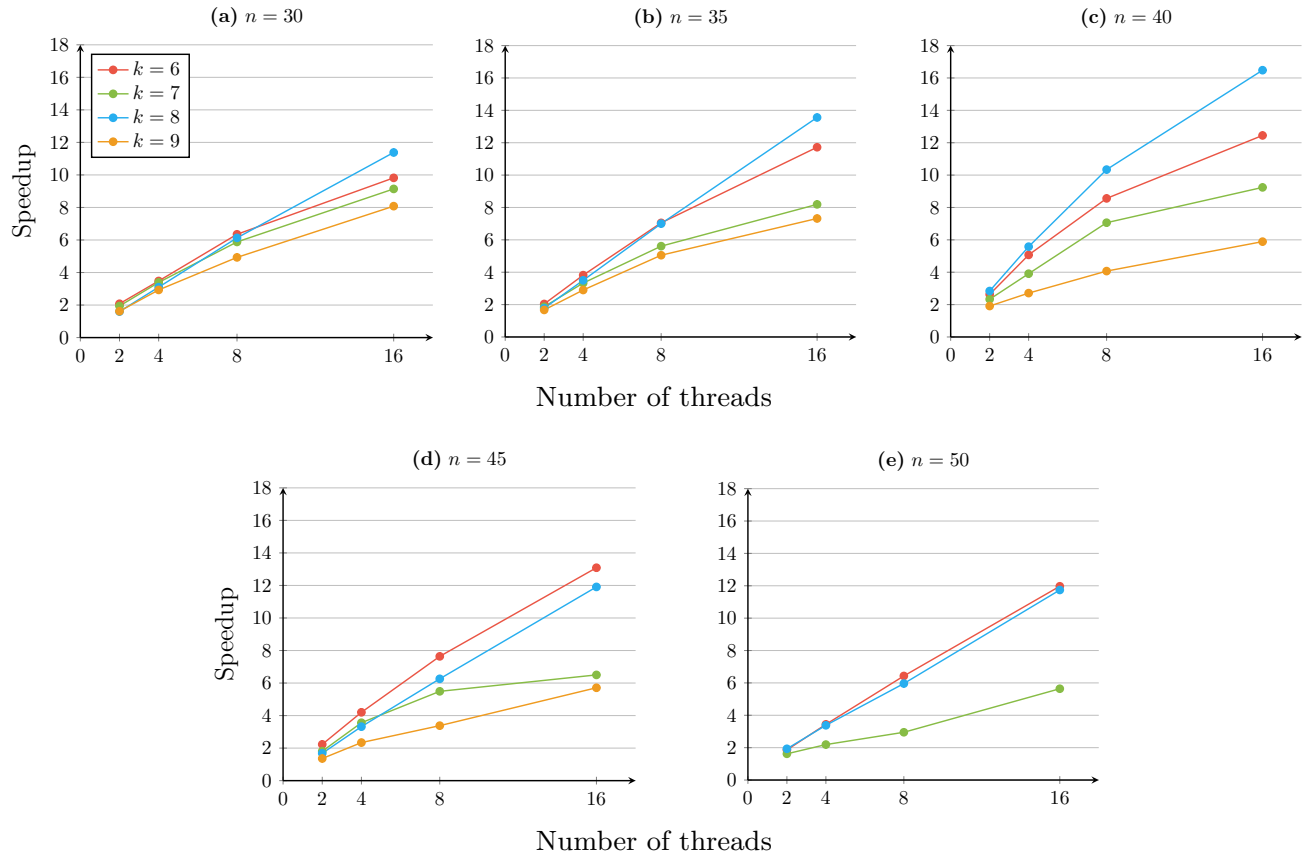


Fig. 5 Scalability of the PIRNP algorithm

5. The running time of the IRNP (also PIRNP) algorithm is affected by the values of the elements in the multiset. Figure 4 shows the variation of the running time when $n = 30$ and $k = 8$ and also when $n = 30$ and $k = 9$ for 50 instances.

5.4 Scalability

The scalability of a parallel algorithm on a parallel architecture is a measure of how much speedup the parallel algorithm could achieve when the number of processors increases [26].

Table 5 illustrates the speedup, S_{nTh}, of the PIRNP algorithm, which is equal to the ratio of the execution time of the IRNP algorithm, T_s, to the execution time of the PIRNP algorithm using nTh threads, T_{nTh}.

$$S_{nTh} = \frac{T_s}{T_{nTh}}. \tag{5}$$

It is clear that the speedup of the PIRNP algorithm increases when the number of threads increases. The average speedups achieved by the PIRNP algorithm are 1.9, 3.5, 6, and 10 using 2, 4, 8, and 16 threads, respectively.

The scalability of the PIRNP algorithm increases when the number of threads increases as illustrated in Fig. 5.

6 Conclusion and future work

In this paper, we discussed the problem of multi-way number partitioning that aims to divide a multiset of n elements into k subsets such that the largest subset sum is minimized. Many sequential algorithms are developed for solving this problem. We parallelize one of these algorithms that is based on the recursive principle of optimality on a multicore system. The idea of parallelization is based on dividing the search space into chuncks and each chunck is distributed to the available threads to find an optimal solution.

The experimental results on different values of n, k, and number of threads demonstrate that the proposed parallel algorithm improves the running time of the sequential algorithm by 90%, on average, and the algorithm achieves sublinear speedup.

A future work for this research could be to study one of the following questions:

1. What is the effect of parallelizing the ESS algorithm and other approximation algorithms used in the PIRNP algorithm?
2. How to use GPU (Graphics Processing Unit) on MWNP?
3. What is the effect of using high-performance systems on the Moffitt algorithm (an exact algorithm) and CIW (a parametric algorithm)?

Acknowledgements The authors would like to thank Dr. Ethan L. Schreiber for providing us with the data set he used in his work and the source code of the mentioned sequential algorithms.

References

1. Sharma PS, Kumar S, Gaur MS, Jain V (2022) A novel intelligent round robin cpu scheduling algorithm. Int J Inform Technol 14:1475–1482
2. Khamis SM, Reda NM, Zakaria W (2022) Combining range-suffrage and sort-mid algorithms for improving grid scheduling. J Supercomput 78:3072–3090
3. Deepakraj D, Raja K (2021) Markov-chain based optimization algorithm for efficient routing in wireless sensor networks. Int J Inform Technol 13:897–904
4. Kenawy TG, Abdel-Rahman MH, Bahig HM (2022) A fast longest crossing-plain preserving common subsequence algorithm. Int J Inform Technol 14:3019–3029
5. Abbas MM, Bahig HM (2016) A fast exact sequential algorithm for the partial digest problem. BMC Bioinform 17(Suppl 19):510
6. Nassr DI, Khamis SM (2021) Applying permutations and cuckoo search for obtaining a new steganography approach in spatial domain. Int J Netw Secur 23(1):67–76
7. Bahig HM, Kotb Y (2019) An efficient multicore algorithm for minimal length addition chains. Comput 8(1):23
8. Bahig HM, Kotb Y (2019) A multicore exact algorithm for addition sequence. J Comput 14(1):79–87
9. Abbas MM, Abouelhoda M, Bahig HM (2012) A hybrid method for the exact planted (l, d) motif: Finding problem and its parallelization. BMC Bioinform 13(Suppl 17):S10
10. Abbas MM, Bahig HM, Abouelhoda M, Mohie-Eldin MM (2014) Parallelizing exact motif finding algorithms on multi-core. J Supercomputer 69(2):814–826
11. Bahig HM, Abbas MM, Mohie-Eldin MM (2017) Parallelizing partial digest problem on multicore system. In: Bioinformatics and biomedical engineering: 5th international work-conference, IWBBIO 2017, Granada, Spain, April 26–28, 2017, Proceedings, Part II 5. pp 95–104
12. Bahig HM, Abbas MM (2018) A scalable parallel algorithm for turnpike problem. J Egypt Math Soc 26:18–26
13. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45(9):1563–1581
14. Karmarker N, Karp RM (1983) The differencing method of set partitioning, Tech. Rep. UCB/CSD-83-113, EECS Department, University of California, Berkeley
15. Korf RE (2009) Multi-way number partitioning. In: Proceedings of the 21st international joint conference on artificial intelligence, San Francisco, CA, USA, pp. 538–543. Morgan Kaufmann Publishers Inc.
16. Korf RE (2011) A hybrid recursive multi-way number partitioning algorithm. In: Proceedings of the 22nd international joint conference on artificial intelligence. pp 591–596
17. Schreiber EL, Korf RE (2014) Cached iterative weakening for optimal multi-way number partitioning. In: Proceedings of the twenty-eighth AAAI conference on artificial intelligence, pp. 2738–2745. AAAI Press
18. Yakir B (1996) The differencing algorithm ldm for partitioning: A proof of a conjecture of karmarkar and karp. Math Oper Res 21(1):85–99
19. Schreiber EL (2014) Optimal multi-way number partitioning. University of California, Los Angeles
20. Schreiber EL, Korf RE, Moffitt MD (2018) Optimal multi-way number partitioning. ACM 65(4):24:1-24:61
21. Horowitz E, Sahni S (1974) Computing partitions with applications to the knapsack problem. J ACM 21(2):277–292
22. Schroeppel R, Shamir A (1981) A $T = O(\frac{2n}{2}), S = O(\frac{2n}{4})$ Algorithm for Certain NP-Complete Problems, SIAM J Comput vol 10, pp 456–464
23. Moffitt MD (2013) Search strategies for optimal multi-way number partitioning. In: Proceedings of the 23rd international joint conference on artificial intelligence, IJCAI '13, pp 623-629. AAAI Press
24. Korf RE, Schreiber EL, Moffitt M D (2014) Optimal sequential multi-way number partitioning. In: International Symposium on Artificial Intelligence and Mathematics, 2014
25. Korf RE (1998) A complete anytime algorithm for number partitioning. Artificial Intell 106(2):181–203
26. Kumar V, Gupta A (1994) Analyzing scalability of parallel algorithms and architectures. J Parallel Distributed Comput 22(3):379–391