# Heuristic Algorithms and Scatter Search for the Cardinality Constrained $P||C_{\max}$ Problem

Mauro Dell'Amico

*DISMI, Università di Modena e Reggio Emilia, viale Allegri 13, 42100 Reggio Emilia, Italy*

Manuel Iori, Silvano Martello

*DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy*

E-mails: dellamico@unimore.it, miori@deis.unibo.it, smartello@deis.unibo.it

Revised version March 3, 2004

Corresponding author:
Prof. Silvano Martello
DEIS, University of Bologna
Viale Risorgimento 2
40136 Bologna
Italy
Phone: +39 051 2093022
Fax: +39 051 2093073
e-mail: smartello@deis.unibo.it

**Abstract**

We consider the generalization of the classical $P||C_{\max}$ problem (assign $n$ jobs to $m$ identical parallel processors by minimizing the makespan) arising when the number of jobs that can be assigned to each processor cannot exceed a given integer $k$. The problem is strongly NP-hard for any fixed $k > 2$. We briefly survey lower and upper bounds from the literature. We introduce greedy heuristics, local search and a scatter search approach. The effectiveness of these approaches is evaluated through extensive computational comparison with a depth-first branch-and-bound algorithm that includes new lower bounds and dominance criteria.

**Key words**: Scheduling, parallel processors, cardinality constraint, scatter search.

Given $n$ jobs, each characterized by a processing time $p_j$ ($j = 1, \ldots, n$), and $m$ identical parallel processors, each of which can process at most one job at a time, consider the problem of assigning each job to a processor so that the maximum completion time of a job (*makespan*) is minimized. The problem is denoted as $P||C_{\max}$ in the three field notation by Graham et al. [12] and is known to be strongly NP-hard. The problem can also be seen as the 'dual' of another famous combinatorial optimization problem that will be also considered in the following: The *Bin Packing Problem*, calling for the partitioning of a given set of $n$ items, each having an associated weight $p_j$, into the minimum number of subsets (*bins*) such that the total weight in each subset does not exceed a given *capacity* $c$. It is then clear that, by determining the minimum $c$ value such that a bin packing instance has an $m$-subset solution, we also solve the associated $P||C_{\max}$ instance.

In this paper we consider a generalization of $P||C_{\max}$ in which an additional constraint imposes that the number of jobs that can be assigned to a processor is at most $k$, denoted as $P|\#\leq k|C_{\max}$. The problem is strongly NP-hard for any fixed $k > 2$ (see Dell'Amico and Martello [5]), while for $k = 2$ it is solvable in $O(n \log n)$ time by sorting the jobs according to non increasing processing time and assigning job $j$ to processor $j$ for $j = 1, \ldots, m$, and job $m + j$ to processor $m - j + 1$ for $j = 1, \ldots, n - m$. We assume that the processing times $p_j$ are non-negative integers. In order to avoid trivial or infeasible instances, we also assume that $2 \leq m$, $2m \leq n$ and that $n \leq mk$.

Possible applications of $P|\#\leq k|C_{\max}$ arise when $m$ processors (e.g., cells of a Flexible Manufacturing System, robots of an assembly line), have to perform $n$ different types of operation. In real world contexts, each processor can have a limit $k$ on the number of different types of operation it can perform, coming, e.g., from the capacity of the cell tool magazine or the number of robot feeders. If it is imposed that all the operations of type $j$ ($j = 1, \ldots, n$) have to be performed by the same processor, and $p_j$ is the total time they require, then $P|\#\leq k|C_{\max}$ models the problem of performing all operations with minimum makespan.

Lower bounds for $P|\#\leq k|C_{\max}$ were presented by Dell'Amico and Martello [5]. The special case arising when $n = mk$, usually denoted as *k-partitioning problem* (*KPP*), was studied by Babel, Kellerer and Kotov [1]. Note that an instance of $P|\#\leq k|C_{\max}$ can be transformed into an instance of $k$-partitioning by adding $n - mk$ dummy jobs with zero processing time.

In Section 1 we review lower bounds from the literature. In Section 2 we present greedy heuristics and in Section 3 a scatter search algorithm with local search procedures. In Section 4 we introduce an enumerative algorithm, together with lower bounds and dominance criteria. Finally, in Section 5, the effectiveness of our approaches is tested through extensive computational experiments performed both on random data sets and real world instances.

# 1 Lower bounds from the literature

Problem $P|\#\leq k|C_{\max}$ can be formally stated as:

$$\min z \tag{1}$$

$$\sum_{j=1}^{n} p_j x_{ij} \leq z \quad (i=1,\ldots,m) \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j=1,\ldots,n) \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} \leq k \quad (i=1,\ldots,m) \tag{4}$$

$$x_{ij} \in \{0,1\} \quad (i=1,\ldots,m; j=1,\ldots,n) \tag{5}$$

where $z$ is the optimum makespan, and $x_{ij}$ takes the value 1 iff job $j$ is assigned to processor $i$. Without loss of generality we assume that the jobs are sorted by non-increasing value of their processing time. Since any lower bound for $P||C_{\max}$ (modeled by (1) – (3) and (5)) is obviously valid for $P|\#\leq k|C_{\max}$, we will both consider bounds adapted from $P||C_{\max}$ and *KPP*, and bounds that explicitly take into account the new constraint.

Dell'Amico and Martello [4, 5] proposed a simple lower bound,

$$L_2 = \max\left(\left\lceil \frac{1}{m}\sum_{j=1}^{n} p_j \right\rceil, \max_j\{p_j\}, p_m + p_{m+1}\right) \tag{6}$$

given by the maximum among the solution value of the continuous relaxation, the largest processing time of a job and the minimum makespan of a processor when no less than $m+1$ jobs have to be scheduled. When $n > m(k-1)$, the bound was strengthened by observing that at least one machine must process $k$ jobs among the first (largest) $m(k-1)+1$ ones: By considering the $k$ smallest such jobs we obtain:

$$\widetilde{L}_2 = \max\left(L_2, \sum_{j=(m-1)(k-1)+1}^{m(k-1)+1} p_j\right) \tag{7}$$

We note that, in the special case of *KPP* (where $n = mk$), the latter bound can be further improved by also considering a lower bound on the makespan of the processor that handles the largest job:

$$\widetilde{L}'_2 = \max\left(\widetilde{L}_2, p_1 + \sum_{j=n-k+2}^{n} p_j\right) \tag{8}$$

All the above bounds can be computed in $O(n)$ time (plus $O(n\log n)$ time for item sorting). The scatter search heuristic of Section 3 and the enumerative algorithm of Section 4 make also use of other more complex bounds from the literature, for which we just give an intuitive explanation, referring the reader to the specific papers. In particular:

- $L_3$: This bound was developed by Dell'Amico and Martello [4] for $P||C_{\max}$, and is based on a partition of the jobs, according to their processing time, determined by a threshold value $\overline{p}$. Each $\overline{p}$ value produces a valid lower bound, and $L_3$, the maximum among them, is determined in a time that is a pseudo-polynomial function of an upper bound on the optimum makespan.

- $L_3^k$: Developed for $P|\#\leq k|C_{\max}$ by Dell'Amico and Martello [5], this bound too is based on thresholds and job partitioning, and has pseudo-polynomial time complexity.

- $L_{BKK}$: polynomial time bound proposed by Babel, Kellerer and Kotov [1] for the $k$-partitioning problem, given by the maximum among three bounds obtained from continuous relaxations and considerations related to the famous $LPT$ heuristic for $P||C_{\max}$ (see below, Section 2).

- $L_{HS}$: Consider the associated bin packing instance described in the Introduction. Hochbaum and Schmoys [14] have proposed an approximation algorithm that, for a given capacity $c$, solves, in linear time, a relaxed problem that provides a lower bound $m(c)$ on the number of bins needed in any feasible solution. We then obtain $L_{HS}=\max\{c+1 : m(c)>m\}$, that is computed in psuedo-polinomial time through binary search on $c$.

# 2 Heuristic algorithms

In this section we first describe heuristic algorithms for $P||C_{\max}$, and then heuristics obtained by modifying them so as to handle the cardinality constraint. In the following we denote by $C(i)$ the current completion time of processor $i$, by $k(i)$ the number of jobs currently assigned to $i$, by $L$ the best lower bound value obtained and by $z$ the incumbent solution value.

## 2.1 Heuristic algorithms for $P||C_{\max}$

Many approximation algorithms are available for $P||C_{\max}$ (see, e.g., the surveys by Lawler et al. [17], Hoogeveen, Lenstra and van de Velde [15], Mokotoff [20]).

A very popular approach is the *List Scheduling* (*LS*) approximation algorithm (see Graham [11]), that sequentially assigns the jobs, in some pre-specified order, to the processor $i$ with minimum $C(i)$, without introducing idle times. If we apply *LS* to a job list sorted by non-increasing $p_j$ value, we obtain the so called *Longest Processing Time* (*LPT*) algorithm, which often produces good approximate solutions (see also its probabilistic analysis in Coffman, Lueker and Rinnooy Kan [3]).

A different approach is the *Multi Fit* (*MF*) heuristic (see Coffman, Garey and Johnson [2]) that finds the smallest value $u$ for which an approximate solution to an associate bin packing problem instance uses no more than $m$ bins of capacity $u$.

Another effective $P||C_{\max}$ heuristic is the *Multi Subset* (*MS*) algorithm by Dell'Amico and Martello [4]. Given $n$ items $j$ with weights $p_j$ ($j = 1, \ldots, n$), and a prefixed capacity $c$, the *Subset-Sum Problem* (*SSP*) is to find a subset of the items whose total weight is closest to, without exceeding, $c$. Given a lower bound $L$ on the $P||C_{\max}$ solution value, algorithm *MS* works as follows. At iteration $i$ ($i = 1, \ldots, m$), *MS* solves an SSP on the instance induced by the currently unassigned jobs with capacity $L$, and assigns the resulting job subset to processor $i$. When all the processors have been considered, the residual unassigned jobs, if any, are assigned through the *LPT* heuristic. The SSP instance considered at each iteration can be solved either exactly (in non-polynomial worst-case time, being the problem NP-hard) or heuristically, through the algorithms in Martello and Toth [18, 19].

## 2.2 Heuristic algorithms for $P|\# \leq k|C_{\max}$

We describe here three heuristics for $P|\# \leq k|C_{\max}$, namely algorithms $LPT_k$, $MS_k$ and $MS2_k$, obtained by adapting algorithms for $P||C_{\max}$ so as to handle the cardinality constraint.

Algorithm $LPT_k$ was already introduced in [1]: At iteration $j$ ($j = 1, \ldots, n$), job $j$ (the largest unassigned job) is assigned to the processor $i$ with minimum $C(i)$ value among those satisfying $k(i) < k$. Ties are broken by selecting the largest $k(i)$ value.

We derived algorithm $MS_k$ from algorithm *MS* described in Section 2.1. In the iterative phase, the associated SSP instance is solved by only considering subsets of cardinality not greater than $k$. In the second phase, the residual unassigned jobs are assigned through $LPT_k$. The specialized algorithm for SSP was obtained by adapting algorithm $G^2$ by Martello and Toth [18]. Algorithm $G^2$ is an $O(n^2)$ time heuristic for SSP that selects the best solution among $O(n)$ solutions produced by a greedy algorithm executed on items sets $\{1, \ldots, n\}$, $\{2, \ldots, n\}$, $\{3, \ldots, n\}$, ..., respectively. The greedy algorithm for SSP iteratively considers all the items: The next item is inserted into the current subset if the capacity is not exceeded. In order to adapt it, it is then enough to terminate its execution as soon as the cardinality limit has been reached.

Algorithm $MS2_k$ is based on partial enumeration and algorithm $MS_k$ above. We start by generating the first $\ell$ levels of our branch-and-bound algorithm (see below, Section 4): The leaves of the resulting branch-decision tree represent all non-dominated solutions involving the $\ell$ largest jobs. The current lower bound value $L$ is then possibly improved by the smallest lower bound associated with a leaf. For each leaf, we complete the associated partial solution through an adaptation of $MS_k$ that:

(i) only uses items $\{\ell + 1, \ldots, n\}$;

(ii) at each iteration of the first phase (i.e., at each solution of an induced SSP solution), decreases the available capacity and the maximum cardinality of the current processor $i$ by the total weight $C(i)$ and number of jobs $k(i)$, respectively, currently assigned to $i$ in the leaf solution;

(iii) assigns the residual unassigned jobs through $LPT_k$.

The best complete solution obtained from a leaf is finally selected. In our implementation, the value $\ell = 5$ was adopted, based on the outcome of computational experiments.

# 3 Scatter search

This metaheuristic technique derives from strategies proposed in the Sixties for combining decision rules and constraints (see Glover [6, 7]), and was successfully applied to a large set of problems (see, e.g., Glover [8, 9]). The basic idea (see Laguna [16], Glover, Laguna and Martí [10]) is to create a set of solutions (the *reference set*), that guarantees a certain level of "quality" and of "diversity". The iterative process consists in selecting a subset of the reference set, in combining the corresponding solutions, through a tailored strategy, in order to create new solutions, and in improving them through local optimization algorithms. The process is repeated, with the use of diversification techniques, until certain stopping criteria are met.

## 3.1 Local optimization algorithms

In this section we introduce the local search algorithms used within our scatter search approach. All the algorithms receive in input a feasible solution, with processors sorted by non-increasing $C(i)$ value.

Procedure *MOVE*: For each processor $i$, in order, let $j$ be the largest job currently assigned to $i$, and execute the following steps:

a. find the first processor $h > i$, if any, such that $k(h) < k$ and $C(h) + p_j < C(i)$, and move job $j$ to $h$;

b. if no such $h$ exists, let $j$ be the next largest job of $i$, if any, and go to a.

As soon as a move is executed, the procedure is re-started, until no further move is possible.

Procedure *EXCHANGE*: For each processor $i$, in order, let $j$ be the largest job currently assigned to $i$, and execute the following steps:

a. find the first processor $h > i$, if any, such that there is a job $q$, currently assigned to $h$, satisfying $p_q < p_j$ and $C(h) - p_q + p_j < C(i)$, and interchange $j$ and $q$;

b. if no such $h$ exists, let $j$ be the next largest job of $i$, if any, and go to a.

As soon as an exchange is executed, the procedure is re-started, until no further exchange is possible.

Procedure *REOPT*: For each processor $i$ satisfying $L \leq C(i) < z$, in order, execute the following steps:

a. remove from the instance the jobs currently assigned to $i$;

b. solve the reduced instance, with $m-1$ processors, through $LPT_k$ followed by *MOVE* and *EXCHANGE*;

c. complete the solution by re-assigning to $i$ the removed jobs.

In addition, the following two improvement procedures are used for *KPP* instances.

Procedure $MIX_k$: This algorithm adopts a sort of dual strategy with respect to $MS_k$ (see Section 2.2). It receives in input a feasible solution and two parameters, $\overline{n}$ and $\overline{k}$ ($1 < \overline{n} < n$, $1 < \overline{k} \leq k$), and creates a new solution as follows:

1. assign the first $\overline{n}$ jobs as in the input solution;

2. sort the processors according to non-increasing $C(i)$ value;
   **for** $i := 1$ **to** $m$ **do**

   $k' := k - k(i)$;

   **if** $k' > \overline{k}$ **then**

   assign to $i$ the smallest $k' - \overline{k}$ unassigned jobs;

   $k' := \overline{k}$;

   **end if**;

   find, through complete enumeration, a set $S$ of $k'$ unassigned jobs, such that $\sum_{s \in S} p_s + C(i)$ is:

   (a) closest to, without exceeding, $L$, if such an $S$ exists;

   (b) closest to $L$ otherwise;

   **end for**

Based on our computational experiments, we adopted the values $\overline{k} = 4$ and $\overline{n} = \max\{m, n - 2m\}$ (but $\overline{n} = \max\{m, n - 4m\}$ at the first scatter search iteration).

Procedure $MIX2_k$: This is a variant of $MIX_k$ in which step 1 is replaced by:

1. assign the first $\widetilde{k}$ jobs of each processor as in the input solution;

where $\widetilde{k}$ is a given parameter for which we adopted the value $\widetilde{k} = \max\{0, (k-2)\}$ (but $\widetilde{k} = \max\{0, (k-4)\}$ at the first scatter search iteration).

It is not difficult to adapt both $MIX_k$ and $MIX2_k$ to non-*KPP* instances, although our computational experiments only showed good results for the *KPP* case.

## 3.2 Scatter search strategy

We first outline the main elements of our scatter search approach and then give the details of the various steps.

1. Randomly generate a starting set $P$ of solutions. Improve each of them through *intensification*.

2. Associate with each solution a positive integer value (*fitness*) that describes its "quality".

3. Create a reference set $R = R_\alpha + R_\beta$ of distinct solutions by including in $R_\alpha$ the $\alpha$ solutions of $P$ with highest fitness, and in $R_\beta$ the $\beta$ solutions of $P \backslash R_\alpha$ with highest *diversity*.

4. Evolve the reference set $R$ through the following steps:

   a. *Subset generation*: generate a family $F$ of subsets of $R$.

   b. **while** $F \neq \emptyset$ **do**
      *Combination*: extract a subset from $F$ and apply
      the combination method to obtain a solution $s$;
      improve $s$ through *intensification*;
      execute the *reference set update* on $R$
      **endwhile**;

   c. **if** *stopping criteria* are not met **then go to** a.

In our implementation, the initial set $P$ has size 80, while the reference set $R$ has size 15, with $\alpha = 8$ and $\beta = 7$. The other main features of the approach are:

a. *Intensification*. It consists in executing, in sequence: $MIX_k$ and $MIX2_k$ (only for *KPP* instances), *REOPT*, *MOVE* and *EXCHANGE*.

b. *Fitness*. In order to highlight the differences between solutions that have very close values, we use a fitness function, instead of the value of the solution. This allows us to obtain a less flat search space, and directs the search towards more promising areas. If $z(s)$ is the value of solution $s$, the correspondent fitness is defined as

$$f(s) = z(s)/(z(s) - L) \tag{9}$$

where $L$ denotes the best lower bound value obtained so far.

c. *Diversity*. The diversity of a solution from those in the current reference set is evaluated by considering the $2m$ jobs with larger processing time. For a solution $s$, let $y_{sj}$ $(j = 1, \ldots, 2m)$ be the processor job $j$ is allocated to. The diversity of $s$ is then

$$d(s) = \min_{r \in R} |\{j \in \{1, \ldots, 2m\} : y_{sj} \neq y_{rj}\}| \tag{10}$$

d. *Subset generation.* We adopted the multiple solution method (see, e.g., Glover, Laguna and Martí [10]), that generates:

   i. all 2-element subsets;

   ii. the 3-element subsets that are obtained by augmenting each 2-element subset to include the best solution not already belonging to it;

   iii. the 4-element subsets that are obtained by augmenting each 3-element subset to include the best solution not already belonging to it;

   iv. the $i$-element subsets (for $i = 5, \ldots \alpha + \beta$) consisting of the best $i$ elements.

e. *Combination.* For a given subset $S$, we define an $m \times n$ fitness matrix $F$ with $F_{ij} = \sum_{s \in S(i,j)} f(s)$, where $S(i,j) \subseteq S$ is the set of solutions where job $j$ is assigned to processor $i$ and $f(s)$ is defined as in (9). We then select the best among three solutions, each created through a random process that, for $j^* = 1, \ldots, n$, assigns job $j^*$ to processor $i^*$ with probability $F(i^*, j^*) / \sum_{i=1}^{m} F(i, j^*)$: if processor $i^*$ now has $k$ jobs assigned, we set $F(i^*, j) = 0$ for $j = 1, \ldots, n$ (so $i^*$ is not selected at the next iterations). If for the current job $j^*$ we have $F(i, j^*) = 0$ for all $i$, the job is assigned to the processor with minimum completion time $C(i)$ among those with less than $k$ jobs assigned.

f. *Reference set update.* In order to evolve the reference set $R$ by maintaining a good level of quality and diversity, we adopted the *dynamic reference set update* (see, e.g., Glover, Laguna and Martí [10]). A new solution immediately enters $R$ if its quality is better than that of the worst solution of $R_\alpha$, or if its diversity is greater than that of the less different solution of $R_\beta$. Solutions that are equal to others already in $R$ are not allowed to enter under any condition.

g. *Stopping criteria.* The scatter search is halted if: (i) the incumbent solution has value equal to lower bound $L$; or (ii) no reference set update occurs at Step 4.; or (iii) Step 4. has been executed 10 times.

# 4 The enumeration algorithm

The results of the previous sections have been embedded into a depth-first branch-and-bound algorithm, derived from that developed by Dell'Amico and Martello [4] for $P||C_{\max}$. At level $j$ of the branch-decision tree, let $M_j$ be the subset of processors $i$ satisfying $k(i) < k$ and $C(i) + p_j < z$: $|M_j|$ nodes are then generated, by assigning job $j$ to the processors in $M_j$. In order to avoid the generation of equivalent solutions, only processors with different $C(i)$ or $k(i)$ value are considered during the branching phase.

At the root node, the algorithm computes the overall lower bound (see Section 1)

$$L = \max\{\overline{L}_2, L_3, L_3^k, L_{BKK}, L_{HS}\}$$

where $\overline{L}_2$, according to the specific instance, denotes $L_2$ or $\widetilde{L}_2$ or $\widetilde{L}'_2$ (see (6)–(8)). In addition, heuristics $LPT$, $MS_k$ and $MS2_k$ (with possible improvement of $L$, see Section 2.2) are executed, followed by the Scatter Search of Section 3.

At each node other than the root, three lower bounds are computed, in sequence, for the current instance: A modified continuous bound $LC$, lower bound $L3$ and lower bound $L3^k$. Since at any intermediate node a partial solution has been already defined, the lower bounding procedure is applied to the remaining sub-instance by excluding the processors with $k(i) = k$ or $C(i) + p_n \geq z$, and by taking into account the fixed decisions as follows.

Lower bound $L3$ is locally computed as in [4]. For $L3^k$, let $\hat{\imath} = \arg\min\{C(h)\}$. We first remove from the instance all the assigned jobs. Then we add, for each processor, a dummy job $j$ with processing time $\widetilde{p}_j = C(i) - C(\hat{\imath})$ (excluding dummy jobs with $\widetilde{p}_j = 0$). The cardinality limit $k$ is then decreased by the minimum number of jobs completely executed on any processor in time interval $[0, C(\hat{\imath})]$. (In order to minimize the resulting $k$ value, it is convenient to sort, for each processor, the scheduled jobs according to non-decreasing processing time.) The bound of the node is then given by $C(\hat{\imath})$ plus the lower bound computed on the instance induced by the dummy and the unassigned jobs.

For the modified continuous bound $LC$, the fixed decisions are taken into account by: (i) assigning to all processors $i$ with $k(i) = k - 1$ the longest unassigned job $j$ such that $C(i) + p_j < z$, and excluding these processors and jobs; (ii) computing the continuous bound $LC = \lceil \sum_{j \in J} p_j / \overline{m} \rceil$, where $J$ is the set of unassigned and non-excluded jobs and $\overline{m}$ is the number of non-excluded processors.

The enumeration algorithm also includes dominance considerations. Three dominance criteria were introduced in [4] for $P||C_{\max}$. One of these (Criterion 1: If $p_j = p_{j+1}$ and $j$ is currently assigned to processor $h$, at level $j+1$ only processors $i$ satisfying $C(i) \geq C(h) - p_j$ are considered for the assignment of job $j+1$) directly applies to $P|\#\leq k|C_{\max}$. The other two were adapted to the cardinality constraint, and are as follows. Let $I$ denote the current set of processors with $k(i) < k$:

> Criterion 2: At level $j$, let $\overline{n} = n - j + 1$ be the number of unassigned jobs. If $\overline{n} < |I|$, only the $\overline{n}$ processors of $I$ with smallest $C(i)$ must be considered for the assignment of job $j$.
>
> Criterion 3: At level $n-2$, let $i_{\min}$ (resp. $i_{\text{smin}}$) be the processor of $K$ with minimum (resp. second minimum, if any) $C(i)$. If $k(i_{\min}) = k - 1$ or $|I| = 1$, the optimal completion of the current schedule is the solution produced by the $LPT_k$ rule. Otherwise it is the best between the $LPT_k$ solution and that obtained by assigning job $n-2$ to $i_{\text{smin}}$ and jobs $n-1$ and $n$ to $i_{\min}$.

We finally describe a fathoming criterion adopted, for the $KPP$ instances, at each decision-node, where job $j$ is assigned to processor $i$. If, after such assignment, we have $0 < k(i) \leq k - 2$, we can consider the minimum possible addition $s(i)$ to the processing time of $i$: $s(i) = \sum_{h=n-(k-k(i))+1}^{n} p_h$ (total processing time of the smallest $k - k(i)$ jobs). If $C(i) + s(i) \geq z$, the node can immediately be fathomed. If instead $C(i) + s(i) < z$, consider the previous job $r(i) = n - (k - k(i))$. If $C(i) + s(i) - p_n + p_{r(i)} \geq z$, we know that job $n$

has to be assigned to $i$. Hence, we can fathom the node if there exists a processors $q \neq i$ for which the minimum possible makespan, $C(q) + s(q) - p_n + p_{r(q)}$, is no less than $z$.

# 5 Computational experiments

The algorithms of the previous sections have been coded in $C$++ and experimentally tested, on a DELL Dimension 8250 with Intel Pentium IV at 2.4 GHz running under a Windows 2000 operating system, both on random instances and on real world instances.

## 5.1 Random instances

We used fifteen classes of randomly generated instances. The first nine classes were already adopted in the computational experiments in [5]:

*Classes* 1, 2 and 3: uniform distribution with $p_j$ in $[10, 1000]$, $[200, 1000]$ and $[500, 1000]$, respectively;

*Classes* 4, 5 and 6: exponential distribution of average value $\mu = 25$, $\mu = 50$ and $\mu = 100$, respectively, by disregarding non-positive values;

*Classes* 7, 8 and 9: normal distribution of average value $\mu = 100$, and standard deviation $\sigma = 33$, $\sigma = 66$ and $\sigma = 100$, respectively, by disregarding non-positive values;

In order to investigate more challenging problems, six additional classes were adopted:

*Classes* 10, 11 and 12: $KPP$ instances obtained through uniform distribution with $p_j$ in $[500, 10000]$, $[1000, 10000]$ and $[1500, 10000]$, respectively;

*Classes* 13, 14 and 15: "perfect packing" $KPP$ instances with $z = \sum_{j=1}^{n} p_j/m$, and $z = 1000$, $z = 5000$ and $z = 10000$, respectively. These were obtained by uniformly randomly splitting, for each processor, the segment $[1, z]$ into $k$ segments.

For Classes 1–9, the code was tested with the values $m = (3, 4, 5, 10, 20, 40, 50)$, $n = (10, 25, 50, 100, 200, 400)$ and $k = (3, 4, 5, 10, 20, 40, 50)$. For Classes 10–15, the values were $m = (8, 10, 13, 15, 18, 20, 25, 30)$ and $k = (3, 4, 5, 10, 15, 20, 25)$, with $n = mk \leq 400$. In order to avoid trivial instances, we only considered those satisfying $n > 2m$, $n/m \leq k \leq n/2$ and $mk \leq 4n$. For each triple ($n$, $m$, $k$) 10 instances were generated, hence, in total, 720 instances for each class 1–9, and 490 instances for each class 10–15.

Table 1 presents the overall performance of all algorithms over each class. We evaluated the separate performances of the three initial heuristics ($LPT_k$, $MS_k$ and $MS2_k$), the performance of the scatter search (*Scatter 0* and *Scatter 1*) and that of the enumeration algorithm ($B\&B$). The scatter search was evaluated both when executed from scratch, i.e., with the first reference set containing only random solutions (*Scatter 0*) and when normally executed, i.e., by receiving in input the best solution found by the initial heuristics (*Scatter 1*). The enumeration algorithm was executed with a limit of 10 000 backtrackings. The execution time was not a problem: the maximum CPU time required for the complete execution (lower bounds, initial heuristics, *Scatter 1* and $B\&B$) on any instance was less

than four minutes. Let $z_A$ be the value of the solution found by an algorithm $A$, and $z$ the best solution value obtained. For each algorithm $A$ and for each class we give:

- #best = number of times in which $z_A = z$;
- #opt = number of times in which $z_A = z$ and $z$ was proved to be optimal;
- #missed = number of times in which $z_A > z$ and $z$ was proved to be optimal;
- %gap = average percentage gap. For each instance, the gap was computed as $100(z_A - z)/z$ if $z$ was proved to be optimal, or as $100(z_A - L)/L$ otherwise.

The performance of the scatter search is very good, especially when executed after the initial heuristics. The quality of the approach is also proved by the fact that its performance deteriorates very little if it is executed from scratch.

The results obtained are also represented in Figures 1–5, where white (resp. dashed) bars represent the average percentage values of *#best* (resp. *#opt*) for groups of three similar classes. Exponentially distributed processing times (Classes 4–6) produce the easiest problems, whereas Classes 1–3 and 7–9 are more difficult. The high percentage of optimal solutions found proves however that both the heuristics and the lower bound perform very well for Classes 1–9. The $KPP$ instances (Classes 10–15) are the hardest ones: The number of proved optimal solutions is small, especially for Classes 10–12, for which the lower bound has a poor performance.

The simple heuristics $LPT_k$, $MS_k$ and $MS2_k$ have acceptable performances for Classes 1–9, but give very bad results for the $KPP$ instances. The total number of best solutions found by $MS2_k$ is larger than that found by $LPT_k$, but its average percentage error is always much higher. This can be explained by observing that $MS2_k$, at each leaf, tries to obtain a solution of value equal to the lower bound: when no leaf succeeds in assigning all items, the completion produced by $LPT_k$ can be quite bad. The Scatter Search, even if executed from scratch, always outperforms the other heuristics, both with respect to the number of optimal solutions and to the percentage gap.

Tables $2-6$ present in detail the results of the overall algorithm for all classes. For each value of $n$, $m$ and $k$, column *opt* gives the number of optimal solutions found (out of ten instances), column $\%g$ the average percentage gap, column $\%g_M$ the maximum percentage gap, and columns $t$ and $t_M$ the average and maximum elapsed CPU time. In addition, for each value of $m$, there is a row summarizing the average results. The final row of each table gives the overall average on all the instances of the class.

We can observe that larger values of $n$ or $k$ generally give easier instances. Indeed, the initial heuristics tend to produce much better solutions in these cases. No immediate relation is observed instead between the value of $m$ and the difficulty of the instances. Increasing the values of the processing times gives in general easier instances for Classes 1–12, but harder instances for the "perfect packing" $KPP$ case.

Worth is noting that the average and maximum gap are very small for all instances, and in the great majority of cases they are below 1%.

## 5.2 Real world instances

As mentioned in the Introduction, $P|\#\leq k|C_{\max}$ finds applications in robotized assembly lines. Hillier and Brandeau [13] studied an operation assignment problem arising from a Printed Circuit Board (*PCB*) assembly process inspired by an application at Hewlett-Packard (HP). They experimented their Lagrangian heuristic on four data sets based on real instances provided by HP. Each data set is characterized by

$b$ = number of different board types;

$c$ = number of component types;

$d_i$ = demand of boards of type $i$ ($i = 1, \ldots, b$);

$\nu_{ij}$ = number of components of type $j$ to be placed on a board of type $i$;

$p$ = processing time for placing a component (of any type);

$\overline{k}$ = maximum number of component types that can be assigned to any processor.

For each *PCB* data set, we constructed two $P|\#\leq k|C_{\max}$ data sets as follows:

PCB$_1$: for each board type $i$ ($i = 1, \ldots, b$) and component type $j$ ($j = 1, \ldots, c$) we define a job with processing time $p\nu_{ij}d_i$;

PCB$_2$: for each component type $j$ ($j = 1, \ldots, c$) we define a job with processing time $p\sum_{i=1}^{b}(\nu_{ij}d_i)$.

Worth is mentioning that the values of $n$ obtained in this way are considerably higher than those tested on random instances (see Table 7). For each data set, we solved the three instances obtained by setting $k = \lceil n/m \rceil$, i.e., the minimum value for which a feasible solution exists, and $m = (5, 10, 20)$. (We also attempted higher values of $k$ without observing relevant variations.)

Table 7 presents the results obtained on the 24 resulting instances by all the considered algorithms. For each value of $n$, $m$ and $k$, column $L$ gives the best lower bound value, column $z$ the best solution value obtained, column *opt* the value 1 (resp. 0) if the solution of value $z$ was (resp. was not) proved to be optimal. The six next columns give, for each algorithm, the percentage gap between the solution value found and $z$ (if $z$ is optimal) or $L$ (otherwise). The last column gives the elapsed CPU time of the overall algorithm. For each data set, the final row summarizes the average results. The overall average is given in the last row.

The table shows that the simplest heuristic, $LPT_k$, has a very good performance, by far dominating that of $MS_k$ and $MS2_k$. The performance of the scatter search is excellent, outperforming the percentage error of $LPT_k$ by two orders of magnitude, both if executed from scratch and starting from the best heuristic solution. The branch-and-bound algorithm could never improve a scatter search solution, indicating that, for these instances, it

is difficult to prove the optimality of a solution. Allowing more than 10 000 backtrackings did not produce improvements.

In conclusion, the overall performance of the proposed scatter search algorithm is very satisfactory both on random instances and real world data sets.

# Acknowledgements

We thank the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) and the Consiglio Nazionale delle Ricerche (CNR), Italy, for the support given to this project. The computational experiments have been executed at the Laboratory of Operations Research of the University of Bologna (LabOR). We thank two anonymous referees for constructive comments that considerably improved this presentation. Thanks are also due to Margaret Brandeau and Mark Hillier for providing the real world instances used in Section 5.2.

# References

[1] L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem. *Mathematical Methods of Operations Research*, 47:59–82, 1998.

[2] E.G. Coffman, M.R. Garey, and D.S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978.

[3] E.G. Coffman, G.S. Lueker, and A.H.G. Rinnooy Kan. Asymptotic methods in the probabilistic analysis of sequencing and packing heuristics. *Management Science*, 34:266–290, 1988.

[4] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.

[5] M. Dell'Amico and S. Martello. Bounds for the cardinality constrained $P||C_{\max}$ problem. *Journal of Scheduling*, 4:123–138, 2001.

[6] F. Glover. Parametric combinations of local job shop rules. In *ONR Research Memorandum no. 117*. GSIA, Carnegie Mellon University, Pittsburgh, Pa, 1963.

[7] F. Glover. A multiphase dual algorithm for the zero-one integer programming problem. *Operation Research*, 13(6):879, 1965.

[8] F. Glover. A template for scatter search and path relinking. In J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Lecture Notes in Computer Science*, volume 1363, pages 1–45. Springer-Verlag, 1997.

[9] F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. Wiley, 1999.

[10] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Foundations and advanced designs. In G. C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*. Springer-Verlag, Heidelberg, 2004.

[11] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[12] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[13] M. S. Hillier and M. L. Brandeau. Optimal component assignment and board grouping in printed circuit board manufacturing. *Operations Research*, 46(5):675–689, 1998.

[14] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of ACM*, 34(1):144–162, 1987.

[15] A. Hoogeveen, J.K. Lenstra, and S.L. van de Velde. Sequencing and scheduling. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 181–197. Wiley, Chichester, 1997.

[16] M. Laguna. Scatter search. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 183–193. Oxford University Press, 2002.

[17] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Handbooks in Operations Research and Management Science*, volume 4, pages 445–522. North-Holland, Amsterdam, 1993.

[18] S. Martello and P. Toth. Worst-case analysis of greedy algorithms for the subset-sum problem. *Mathematical Programming*, 28:198–205, 1984.

[19] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990.

[20] E. Mokotoff. Parallel machine scheduling problems: a survey. *Asia-Pacific Journal of Operational Research*, 18:193–242, 2001.

**Table 1. Overall performance of the algorithms for Classes 1–15.**

| *Class* | | $LPT_k$ | $MS_k$ | $MS2_k$ | *Scatter 0* | *Scatter 1* | *B&B* |
|---|---|---|---|---|---|---|---|
| | #best | 81 | 181 | 303 | 651 | 720 | 720 |
| 1 | #opt | 80 | 181 | 301 | 587 | 646 | 646 |
| | #missed | 566 | 465 | 345 | 59 | 0 | 0 |
| | %gap | 1.655 | 8.246 | 6.073 | 0.052 | 0.047 | 0.047 |
| | #best | 46 | 248 | 345 | 662 | 720 | 720 |
| 2 | #opt | 44 | 246 | 344 | 561 | 611 | 611 |
| | #missed | 567 | 365 | 267 | 50 | 0 | 0 |
| | %gap | 2.668 | 8.752 | 6.275 | 0.093 | 0.090 | 0.090 |
| | #best | 64 | 263 | 372 | 700 | 715 | 720 |
| 3 | #opt | 64 | 263 | 371 | 657 | 671 | 676 |
| | #missed | 612 | 413 | 305 | 19 | 5 | 0 |
| | %gap | 2.965 | 5.500 | 3.747 | 0.161 | 0.160 | 0.160 |
| | #best | 572 | 290 | 326 | 720 | 720 | 720 |
| 4 | #opt | 572 | 290 | 326 | 720 | 720 | 720 |
| | #missed | 148 | 430 | 394 | 0 | 0 | 0 |
| | %gap | 0.235 | 3.810 | 3.108 | 0.000 | 0.000 | 0.000 |
| | #best | 474 | 300 | 335 | 720 | 720 | 720 |
| 5 | #opt | 474 | 300 | 335 | 720 | 720 | 720 |
| | #missed | 246 | 420 | 385 | 0 | 0 | 0 |
| | %gap | 0.261 | 3.597 | 2.999 | 0.000 | 0.000 | 0.000 |
| | #best | 381 | 306 | 351 | 720 | 720 | 720 |
| 6 | #opt | 381 | 306 | 351 | 718 | 718 | 718 |
| | #missed | 337 | 412 | 367 | 0 | 0 | 0 |
| | %gap | 0.276 | 3.587 | 2.887 | 0.001 | 0.001 | 0.001 |
| | #best | 37 | 297 | 359 | 719 | 720 | 720 |
| 7 | #opt | 37 | 297 | 359 | 637 | 638 | 638 |
| | #missed | 601 | 341 | 279 | 1 | 0 | 0 |
| | %gap | 3.618 | 8.018 | 5.756 | 0.093 | 0.093 | 0.093 |
| | #best | 86 | 258 | 334 | 712 | 720 | 720 |
| 8 | #opt | 86 | 258 | 333 | 661 | 667 | 667 |
| | #missed | 581 | 409 | 334 | 6 | 0 | 0 |
| | %gap | 2.250 | 6.842 | 4.712 | 0.049 | 0.045 | 0.045 |
| | #best | 140 | 267 | 333 | 715 | 720 | 720 |
| 9 | #opt | 140 | 267 | 333 | 684 | 687 | 687 |
| | #missed | 547 | 420 | 354 | 3 | 0 | 0 |
| | %gap | 1.404 | 5.605 | 4.165 | 0.025 | 0.023 | 0.023 |
| | #best | 1 | 0 | 0 | 400 | 490 | 490 |
| 10 | #opt | 1 | 0 | 0 | 197 | 223 | 223 |
| | #missed | 222 | 223 | 223 | 26 | 0 | 0 |
| | %gap | 1.859 | 12.050 | 8.835 | 0.045 | 0.043 | 0.043 |
| | #best | 1 | 1 | 0 | 418 | 489 | 490 |
| 11 | #opt | 1 | 1 | 0 | 217 | 231 | 231 |
| | #missed | 230 | 230 | 231 | 14 | 0 | 0 |
| | %gap | 1.682 | 12.467 | 9.522 | 0.041 | 0.039 | 0.039 |
| | #best | 1 | 0 | 0 | 412 | 487 | 490 |
| 12 | #opt | 1 | 0 | 0 | 224 | 245 | 246 |
| | #missed | 245 | 246 | 246 | 22 | 1 | 0 |
| | %gap | 1.520 | 12.815 | 9.843 | 0.037 | 0.036 | 0.035 |
| | #best | 17 | 1 | 2 | 478 | 483 | 490 |
| 13 | #opt | 17 | 1 | 2 | 420 | 425 | 432 |
| | #missed | 415 | 431 | 430 | 12 | 7 | 0 |
| | %gap | 1.421 | 13.529 | 12.554 | 0.018 | 0.017 | 0.015 |
| | #best | 0 | 1 | 2 | 470 | 487 | 490 |
| 14 | #opt | 0 | 1 | 2 | 375 | 380 | 380 |
| | #missed | 380 | 379 | 378 | 5 | 0 | 0 |
| | %gap | 1.400 | 13.672 | 12.119 | 0.015 | 0.014 | 0.014 |
| | #best | 0 | 1 | 2 | 462 | 489 | 490 |
| 15 | #opt | 0 | 1 | 2 | 343 | 353 | 353 |
| | #missed | 353 | 352 | 351 | 10 | 0 | 0 |
| | %gap | 1.400 | 13.603 | 11.790 | 0.015 | 0.014 | 0.014 |

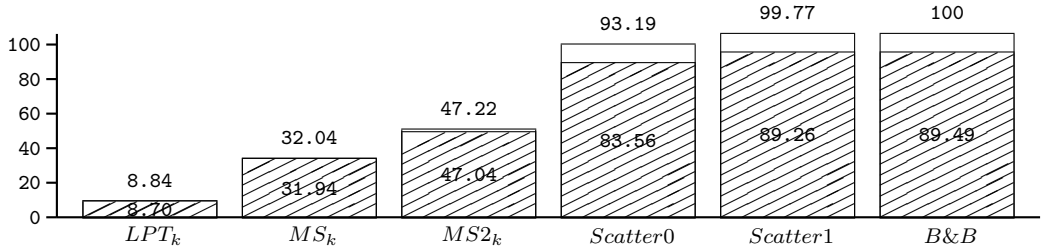

Figure 1: Percentage of optimal solutions for Classes 1–3

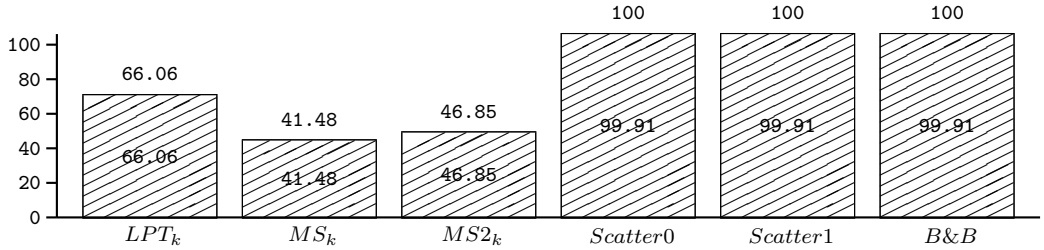

Figure 2: Percentage of optimal solutions for Classes 4–6

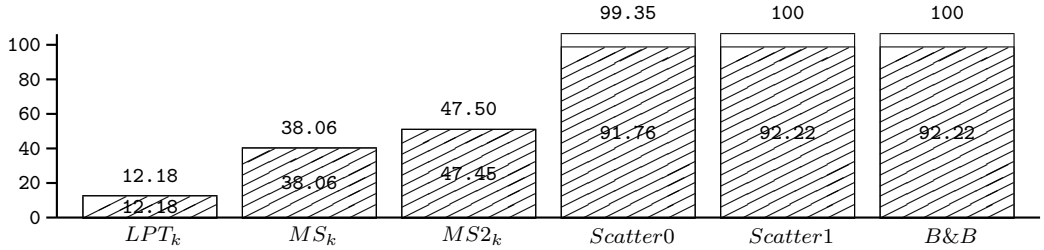

Figure 3: Percentage of optimal solutions for Classes 7–9

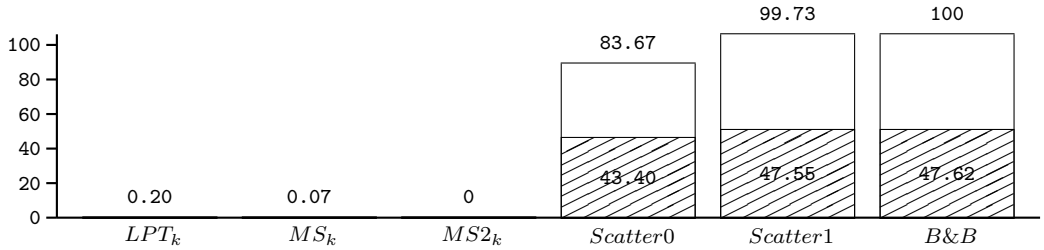

Figure 4: Percentage of optimal solutions for Classes 10–12

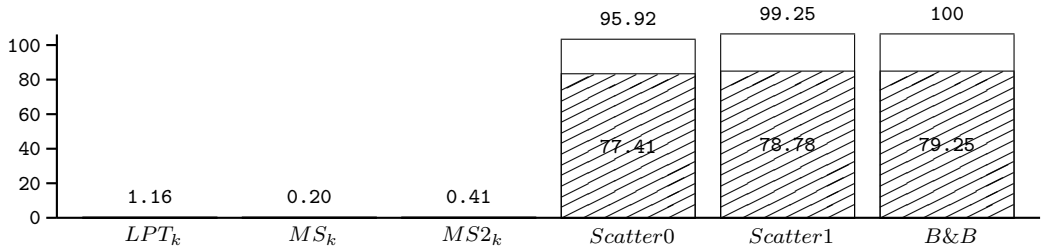

Figure 5: Percentage of optimal solutions for Classes 13–15

**Table 2. Results for Classes 1–3 (uniform distribution).**

| | | | $p_j \in [10, 1000]$ | | | | | $p_j \in [200, 1000]$ | | | | | $p_j \in [500, 1000]$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ |
| 10 | 3 | 4 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.04 | 0.04 | 10 | 0.000 | 0.000 | 0.02 | 0.04 |
| 10 | 3 | 5 | 10 | 0.000 | 0.000 | 0.02 | 0.06 | 10 | 0.000 | 0.000 | 0.03 | 0.05 | 10 | 0.000 | 0.000 | 0.01 | 0.04 |
| 25 | 3 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 3 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 3 | 40 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.04 | 0.10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 3 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| average | | | 10.0 | 0.000 | 0.000 | 0.01 | 0.02 | 10.0 | 0.000 | 0.000 | 0.02 | 0.03 | 10.0 | 0.000 | 0.000 | 0.01 | 0.01 |
| 10 | 4 | 3 | 10 | 0.000 | 0.000 | 0.01 | 0.06 | 10 | 0.000 | 0.000 | 0.03 | 0.05 | 10 | 0.000 | 0.000 | 0.04 | 0.05 |
| 10 | 4 | 4 | 10 | 0.000 | 0.000 | 0.03 | 0.07 | 10 | 0.000 | 0.000 | 0.04 | 0.05 | 10 | 0.000 | 0.000 | 0.04 | 0.05 |
| 10 | 4 | 5 | 10 | 0.000 | 0.000 | 0.04 | 0.09 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.04 | 0.07 |
| 25 | 4 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 4 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 4 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 4 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 4 | 50 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.02 | 0.04 | 10 | 0.000 | 0.000 | 0.01 | 0.03 |
| average | | | 10.0 | 0.000 | 0.000 | 0.01 | 0.03 | 10.0 | 0.000 | 0.000 | 0.01 | 0.02 | 10.0 | 0.000 | 0.000 | 0.02 | 0.03 |
| 10 | 5 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.04 | 10 | 0.000 | 0.000 | 0.01 | 0.04 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 10 | 5 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.04 | 10 | 0.000 | 0.000 | 0.00 | 0.05 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 10 | 5 | 5 | 10 | 0.000 | 0.000 | 0.01 | 0.04 | 10 | 0.000 | 0.000 | 0.00 | 0.04 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 25 | 5 | 5 | 8 | 0.008 | 0.039 | 0.11 | 0.58 | 7 | 0.011 | 0.043 | 0.19 | 0.55 | 9 | 0.003 | 0.026 | 0.05 | 0.46 |
| 25 | 5 | 10 | 9 | 0.004 | 0.039 | 0.07 | 0.64 | 10 | 0.000 | 0.000 | 0.00 | 0.02 | 10 | 0.000 | 0.000 | 0.00 | 0.02 |
| 50 | 5 | 10 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 50 | 5 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 5 | 20 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.01 | 0.01 |
| 100 | 5 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 5 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 5 | 40 | 10 | 0.000 | 0.000 | 0.02 | 0.07 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.01 | 0.03 |
| 200 | 5 | 50 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.02 | 0.02 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| average | | | 9.8 | 0.001 | 0.006 | 0.02 | 0.12 | 9.8 | 0.001 | 0.004 | 0.02 | 0.07 | 9.9 | 0.000 | 0.002 | 0.01 | 0.05 |
| 25 | 10 | 3 | 10 | 0.000 | 0.000 | 0.36 | 0.68 | 8 | 0.169 | 1.617 | 0.40 | 0.87 | 10 | 0.000 | 0.000 | 0.31 | 0.43 |
| 25 | 10 | 4 | 10 | 0.000 | 0.000 | 0.31 | 0.73 | 9 | 0.202 | 2.021 | 0.55 | 0.85 | 10 | 0.000 | 0.000 | 0.39 | 0.57 |
| 25 | 10 | 5 | 10 | 0.000 | 0.000 | 0.19 | 0.66 | 7 | 0.806 | 3.068 | 0.60 | 0.97 | 9 | 0.097 | 0.966 | 0.35 | 0.60 |
| 25 | 10 | 10 | 10 | 0.000 | 0.000 | 0.44 | 0.77 | 8 | 0.490 | 2.828 | 0.63 | 0.99 | 10 | 0.000 | 0.000 | 0.39 | 0.56 |
| 50 | 10 | 5 | 10 | 0.000 | 0.000 | 0.02 | 0.06 | 10 | 0.000 | 0.000 | 0.02 | 0.07 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 50 | 10 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 9 | 0.003 | 0.027 | 0.22 | 2.15 |
| 50 | 10 | 20 | 9 | 0.004 | 0.043 | 0.30 | 2.94 | 9 | 0.003 | 0.032 | 0.22 | 2.20 | 10 | 0.000 | 0.000 | 0.04 | 0.39 |
| 100 | 10 | 10 | 10 | 0.000 | 0.000 | 0.03 | 0.07 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 100 | 10 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 10 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 10 | 20 | 10 | 0.000 | 0.000 | 0.03 | 0.08 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.02 | 0.03 |
| 200 | 10 | 40 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 10 | 50 | 10 | 0.000 | 0.000 | 0.01 | 0.05 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 400 | 10 | 40 | 10 | 0.000 | 0.000 | 0.07 | 0.09 | 10 | 0.000 | 0.000 | 0.06 | 0.08 | 10 | 0.000 | 0.000 | 0.04 | 0.05 |
| 400 | 10 | 50 | 10 | 0.000 | 0.000 | 0.05 | 0.06 | 10 | 0.000 | 0.000 | 0.04 | 0.05 | 10 | 0.000 | 0.000 | 0.03 | 0.05 |
| average | | | 9.9 | 0.000 | 0.003 | 0.12 | 0.42 | 9.4 | 0.111 | 0.638 | 0.17 | 0.41 | 9.9 | 0.007 | 0.066 | 0.12 | 0.33 |
| 50 | 20 | 3 | 6 | 0.251 | 0.746 | 2.17 | 5.36 | 0 | 0.637 | 1.105 | 4.56 | 5.52 | 7 | 0.653 | 2.853 | 4.05 | 6.16 |
| 50 | 20 | 4 | 4 | 0.383 | 1.111 | 2.41 | 4.54 | 0 | 0.685 | 1.697 | 4.51 | 5.22 | 9 | 0.020 | 0.199 | 4.46 | 6.63 |
| 50 | 20 | 5 | 3 | 0.528 | 1.325 | 3.28 | 5.33 | 0 | 0.706 | 1.649 | 4.72 | 7.06 | 8 | 0.873 | 4.836 | 5.21 | 8.21 |
| 50 | 20 | 10 | 2 | 0.448 | 0.819 | 3.88 | 5.71 | 0 | 0.578 | 1.023 | 4.93 | 6.48 | 9 | 0.407 | 4.073 | 4.66 | 6.87 |
| 100 | 20 | 5 | 10 | 0.000 | 0.000 | 0.11 | 0.38 | 10 | 0.000 | 0.000 | 0.07 | 0.23 | 10 | 0.000 | 0.000 | 0.06 | 0.22 |
| 100 | 20 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 5 | 0.013 | 0.027 | 6.31 | 13.49 |
| 100 | 20 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 9 | 0.003 | 0.026 | 1.19 | 11.16 |
| 200 | 20 | 10 | 10 | 0.000 | 0.000 | 0.09 | 0.16 | 10 | 0.000 | 0.000 | 0.07 | 0.12 | 10 | 0.000 | 0.000 | 0.04 | 0.08 |
| 200 | 20 | 20 | 8 | 0.004 | 0.020 | 5.74 | 28.76 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 20 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 400 | 20 | 20 | 10 | 0.000 | 0.000 | 0.11 | 0.16 | 10 | 0.000 | 0.000 | 0.09 | 0.14 | 10 | 0.000 | 0.000 | 0.05 | 0.08 |
| 400 | 20 | 40 | 10 | 0.000 | 0.000 | 0.14 | 0.27 | 10 | 0.000 | 0.000 | 0.06 | 0.15 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 400 | 20 | 50 | 10 | 0.000 | 0.000 | 0.11 | 0.19 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| average | | | 7.9 | 0.124 | 0.309 | 1.39 | 3.91 | 6.9 | 0.201 | 0.421 | 1.46 | 1.92 | 9.0 | 0.151 | 0.924 | 2.00 | 4.07 |
| 100 | 40 | 3 | 5 | 0.207 | 0.516 | 15.44 | 34.18 | 0 | 0.408 | 0.747 | 34.29 | 43.44 | 6 | 1.364 | 3.907 | 46.90 | 55.68 |
| 100 | 40 | 4 | 1 | 0.460 | 0.732 | 26.71 | 33.05 | 0 | 0.385 | 0.543 | 33.77 | 42.56 | 3 | 2.998 | 5.260 | 53.44 | 72.02 |
| 100 | 40 | 5 | 1 | 0.441 | 0.801 | 29.86 | 37.14 | 0 | 0.369 | 0.707 | 30.43 | 34.62 | 3 | 2.951 | 5.407 | 58.03 | 66.20 |
| 100 | 40 | 10 | 1 | 0.476 | 0.701 | 29.01 | 43.52 | 0 | 0.387 | 0.492 | 35.87 | 44.61 | 4 | 2.126 | 4.914 | 62.08 | 77.87 |
| 200 | 40 | 5 | 10 | 0.000 | 0.000 | 0.80 | 2.86 | 10 | 0.000 | 0.000 | 1.91 | 5.24 | 9 | 0.003 | 0.027 | 12.03 | 114.66 |
| 200 | 40 | 10 | 9 | 0.004 | 0.043 | 4.92 | 49.20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 9 | 0.003 | 0.027 | 7.05 | 68.22 |
| 200 | 40 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 9 | 0.003 | 0.027 | 8.37 | 63.04 |
| 400 | 40 | 10 | 10 | 0.000 | 0.000 | 0.40 | 1.13 | 10 | 0.000 | 0.000 | 0.24 | 0.86 | 10 | 0.000 | 0.000 | 0.14 | 0.39 |
| 400 | 40 | 20 | 10 | 0.000 | 0.000 | 4.37 | 34.42 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 400 | 40 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| average | | | 6.7 | 0.159 | 0.279 | 11.15 | 23.55 | 6.0 | 0.155 | 0.249 | 13.65 | 17.14 | 7.3 | 0.945 | 1.957 | 24.80 | 51.81 |
| 100 | 50 | 3 | 9 | 0.057 | 0.569 | 3.91 | 38.80 | 7 | 0.226 | 1.084 | 10.02 | 38.40 | 10 | 0.000 | 0.000 | 0.01 | 0.01 |
| 100 | 50 | 4 | 8 | 0.076 | 0.397 | 6.67 | 34.88 | 7 | 0.157 | 1.007 | 12.18 | 33.64 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 50 | 5 | 9 | 0.030 | 0.302 | 5.68 | 30.07 | 3 | 0.296 | 1.092 | 21.24 | 37.24 | 10 | 0.000 | 0.000 | 0.01 | 0.01 |
| 200 | 50 | 4 | 10 | 0.000 | 0.000 | 0.69 | 5.39 | 10 | 0.000 | 0.000 | 0.20 | 0.62 | 10 | 0.000 | 0.000 | 0.25 | 1.68 |
| 200 | 50 | 5 | 5 | 0.025 | 0.052 | 39.54 | 88.69 | 6 | 0.017 | 0.043 | 30.91 | 95.25 | 9 | 0.003 | 0.034 | 10.88 | 105.53 |
| 200 | 50 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.02 | 9 | 0.003 | 0.034 | 12.63 | 126.31 |
| 400 | 50 | 10 | 10 | 0.000 | 0.000 | 1.62 | 11.74 | 9 | 0.002 | 0.021 | 10.34 | 98.58 | 10 | 0.000 | 0.000 | 0.01 | 0.03 |
| 400 | 50 | 20 | 8 | 0.005 | 0.026 | 26.23 | 129.79 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.01 | 0.03 |
| average | | | 8.6 | 0.024 | 0.168 | 10.54 | 42.42 | 7.8 | 0.087 | 0.406 | 10.61 | 37.97 | 9.8 | 0.001 | 0.008 | 2.98 | 29.20 |
| overall average | | | 9.0 | 0.047 | 0.115 | 3.00 | 8.80 | 8.5 | 0.091 | 0.289 | 3.38 | 7.05 | 9.4 | 0.160 | 0.454 | 4.17 | 11.25 |

**Table 3. Results for Classes 4–6 (exponential distribution).**

| | | | $\mu = 25$ | | | | | $\mu = 50$ | | | | | $\mu = 100$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ |
| 10 | 3 | 4 | 10 | 0.000 | 0.000 | 0.02 | 0.06 | 10 | 0.000 | 0.000 | 0.01 | 0.05 | 10 | 0.000 | 0.000 | 0.02 | 0.06 |
| 10 | 3 | 5 | 10 | 0.000 | 0.000 | 0.01 | 0.07 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.02 | 0.07 |
| 25 | 3 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 3 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 3 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 3 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.01 | 0.02 |
| 10 | 4 | 3 | 10 | 0.000 | 0.000 | 0.01 | 0.06 | 10 | 0.000 | 0.000 | 0.00 | 0.02 | 10 | 0.000 | 0.000 | 0.00 | 0.03 |
| 10 | 4 | 4 | 10 | 0.000 | 0.000 | 0.01 | 0.06 | 10 | 0.000 | 0.000 | 0.01 | 0.06 | 10 | 0.000 | 0.000 | 0.01 | 0.06 |
| 10 | 4 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.01 | 0.04 | 10 | 0.000 | 0.000 | 0.01 | 0.04 |
| 25 | 4 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 4 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 4 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 4 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 200 | 4 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 |
| 10 | 5 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 10 | 5 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 10 | 5 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 25 | 5 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.03 | 0.34 | 9 | 0.014 | 0.136 | 0.11 | 0.66 |
| 25 | 5 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 5 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 50 | 5 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 5 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 5 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.05 |
| 100 | 5 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 5 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 200 | 5 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.00 | 10.0 | 0.000 | 0.000 | 0.00 | 0.03 | 9.9 | 0.001 | 0.011 | 0.01 | 0.06 |
| 25 | 10 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 25 | 10 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.06 | 0.57 |
| 25 | 10 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.06 | 0.56 | 10 | 0.000 | 0.000 | 0.05 | 0.54 |
| 25 | 10 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 10 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 50 | 10 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 50 | 10 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 10 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.02 |
| 100 | 10 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 10 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 10 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 200 | 10 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 200 | 10 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 400 | 10 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.02 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 400 | 10 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.02 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.00 | 10.0 | 0.000 | 0.000 | 0.01 | 0.04 | 10.0 | 0.000 | 0.000 | 0.01 | 0.08 |
| 50 | 20 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 20 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 20 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 20 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 20 | 5 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.05 | 0.26 | 10 | 0.000 | 0.000 | 0.10 | 0.73 |
| 100 | 20 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.06 |
| 100 | 20 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 20 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.02 | 0.16 | 10 | 0.000 | 0.000 | 0.02 | 0.02 |
| 200 | 20 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 200 | 20 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 400 | 20 | 20 | 10 | 0.000 | 0.000 | 0.02 | 0.04 | 10 | 0.000 | 0.000 | 0.03 | 0.04 | 10 | 0.000 | 0.000 | 0.03 | 0.04 |
| 400 | 20 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.02 | 10 | 0.000 | 0.000 | 0.02 | 0.03 |
| 400 | 20 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.04 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.01 | 10.0 | 0.000 | 0.000 | 0.01 | 0.04 | 10.0 | 0.000 | 0.000 | 0.02 | 0.07 |
| 100 | 40 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 40 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 40 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 40 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 40 | 5 | 10 | 0.000 | 0.000 | 0.15 | 1.46 | 10 | 0.000 | 0.000 | 0.23 | 1.31 | 10 | 0.000 | 0.000 | 5.26 | 52.62 |
| 200 | 40 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 9 | 0.024 | 0.236 | 3.54 | 35.36 |
| 200 | 40 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 400 | 40 | 10 | 10 | 0.000 | 0.000 | 0.08 | 0.19 | 10 | 0.000 | 0.000 | 0.21 | 0.89 | 10 | 0.000 | 0.000 | 0.67 | 4.07 |
| 400 | 40 | 20 | 10 | 0.000 | 0.000 | 0.01 | 0.04 | 10 | 0.000 | 0.000 | 0.03 | 0.07 | 10 | 0.000 | 0.000 | 0.04 | 0.14 |
| 400 | 40 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.02 | 0.19 | 10 | 0.000 | 0.000 | 0.03 | 0.18 |
| average | | | 10.0 | 0.000 | 0.000 | 0.02 | 0.17 | 10.0 | 0.000 | 0.000 | 0.05 | 0.25 | 9.9 | 0.002 | 0.024 | 0.95 | 9.24 |
| 100 | 50 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 50 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 50 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 50 | 4 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 50 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 50 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 400 | 50 | 10 | 10 | 0.000 | 0.000 | 0.01 | 0.05 | 10 | 0.000 | 0.000 | 0.01 | 0.08 | 10 | 0.000 | 0.000 | 0.01 | 0.03 |
| 400 | 50 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.04 | 10 | 0.000 | 0.000 | 0.01 | 0.05 | 10 | 0.000 | 0.000 | 0.30 | 1.71 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.01 | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.04 | 0.22 |
| overall average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.03 | 10.0 | 0.000 | 0.000 | 0.01 | 0.06 | 10.0 | 0.001 | 0.005 | 0.14 | 1.35 |

**Table 4. Results for Classes 7–9 (normal distribution).**

| | | | $\mu = 100$, $\sigma = 33$ | | | | | $\mu = 100$, $\sigma = 66$ | | | | | $\mu = 100$, $\sigma = 100$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ |
| 10 | 3 | 4 | 10 | 0.000 | 0.000 | 0.04 | 0.07 | 10 | 0.000 | 0.000 | 0.02 | 0.04 | 10 | 0.000 | 0.000 | 0.03 | 0.05 |
| 10 | 3 | 5 | 10 | 0.000 | 0.000 | 0.04 | 0.06 | 10 | 0.000 | 0.000 | 0.02 | 0.06 | 10 | 0.000 | 0.000 | 0.03 | 0.05 |
| 25 | 3 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 3 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 3 | 40 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 3 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| average | | | 10.0 | 0.000 | 0.000 | 0.01 | 0.02 | 10.0 | 0.000 | 0.000 | 0.01 | 0.02 | 10.0 | 0.000 | 0.000 | 0.01 | 0.02 |
| 10 | 4 | 3 | 10 | 0.000 | 0.000 | 0.03 | 0.07 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.01 | 0.04 |
| 10 | 4 | 4 | 10 | 0.000 | 0.000 | 0.04 | 0.07 | 10 | 0.000 | 0.000 | 0.05 | 0.09 | 10 | 0.000 | 0.000 | 0.04 | 0.08 |
| 10 | 4 | 5 | 10 | 0.000 | 0.000 | 0.04 | 0.07 | 10 | 0.000 | 0.000 | 0.02 | 0.07 | 10 | 0.000 | 0.000 | 0.02 | 0.08 |
| 25 | 4 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 4 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 4 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 4 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 4 | 50 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.02 |
| average | | | 10.0 | 0.000 | 0.000 | 0.02 | 0.03 | 10.0 | 0.000 | 0.000 | 0.01 | 0.03 | 10.0 | 0.000 | 0.000 | 0.01 | 0.03 |
| 10 | 5 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.01 | 0.04 | 10 | 0.000 | 0.000 | 0.00 | 0.03 |
| 10 | 5 | 4 | 10 | 0.000 | 0.000 | 0.01 | 0.06 | 10 | 0.000 | 0.000 | 0.00 | 0.04 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 10 | 5 | 5 | 10 | 0.000 | 0.000 | 0.01 | 0.07 | 10 | 0.000 | 0.000 | 0.01 | 0.10 | 10 | 0.000 | 0.000 | 0.00 | 0.04 |
| 25 | 5 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 25 | 5 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 5 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 50 | 5 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 5 | 20 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.02 |
| 100 | 5 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 5 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 5 | 40 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 200 | 5 | 50 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| average | | | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.00 | 0.02 | 10.0 | 0.000 | 0.000 | 0.00 | 0.01 |
| 25 | 10 | 3 | 10 | 0.000 | 0.000 | 0.48 | 0.73 | 10 | 0.000 | 0.000 | 0.41 | 0.58 | 10 | 0.000 | 0.000 | 0.23 | 0.53 |
| 25 | 10 | 4 | 6 | 0.922 | 4.082 | 0.70 | 1.08 | 9 | 0.394 | 3.943 | 0.49 | 1.03 | 10 | 0.000 | 0.000 | 0.30 | 0.64 |
| 25 | 10 | 5 | 8 | 0.360 | 1.961 | 0.65 | 1.00 | 8 | 0.425 | 2.545 | 0.52 | 0.93 | 10 | 0.000 | 0.000 | 0.35 | 0.89 |
| 25 | 10 | 10 | 6 | 0.841 | 2.672 | 0.76 | 1.20 | 10 | 0.000 | 0.000 | 0.53 | 0.71 | 10 | 0.000 | 0.000 | 0.51 | 0.94 |
| 50 | 10 | 5 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 50 | 10 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 50 | 10 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 10 | 10 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| 100 | 10 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 10 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 10 | 20 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.01 | 0.01 |
| 200 | 10 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.01 | 0.01 |
| 200 | 10 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.01 | 0.01 |
| 400 | 10 | 40 | 10 | 0.000 | 0.000 | 0.03 | 0.05 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.02 | 0.03 |
| 400 | 10 | 50 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.01 | 0.02 |
| average | | | 9.3 | 0.142 | 0.581 | 0.18 | 0.28 | 9.8 | 0.055 | 0.433 | 0.13 | 0.22 | 10.0 | 0.000 | 0.000 | 0.10 | 0.21 |
| 50 | 20 | 3 | 3 | 0.444 | 1.172 | 3.91 | 5.02 | 3 | 0.371 | 1.056 | 3.20 | 4.51 | 5 | 0.303 | 0.880 | 2.20 | 5.65 |
| 50 | 20 | 4 | 2 | 0.626 | 2.335 | 4.92 | 9.72 | 5 | 0.221 | 0.787 | 2.37 | 5.15 | 6 | 0.296 | 1.079 | 2.31 | 5.05 |
| 50 | 20 | 5 | 3 | 0.317 | 0.787 | 4.03 | 5.93 | 2 | 0.514 | 0.823 | 3.83 | 6.00 | 7 | 0.217 | 0.901 | 2.07 | 5.58 |
| 50 | 20 | 10 | 4 | 0.323 | 0.797 | 3.95 | 6.51 | 2 | 0.450 | 0.844 | 3.97 | 5.42 | 2 | 0.462 | 0.990 | 3.91 | 5.39 |
| 100 | 20 | 5 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.02 | 0.07 |
| 100 | 20 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.01 |
| 100 | 20 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 20 | 10 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.03 | 0.03 | 10 | 0.000 | 0.000 | 0.03 | 0.06 |
| 200 | 20 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.02 | 0.03 | 10 | 0.000 | 0.000 | 0.02 | 0.04 |
| 200 | 20 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 400 | 20 | 20 | 10 | 0.000 | 0.000 | 0.04 | 0.06 | 10 | 0.000 | 0.000 | 0.04 | 0.05 | 10 | 0.000 | 0.000 | 0.04 | 0.05 |
| 400 | 20 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.03 | 0.05 | 10 | 0.000 | 0.000 | 0.03 | 0.05 |
| 400 | 20 | 50 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.03 | 0.05 | 10 | 0.000 | 0.000 | 0.03 | 0.04 |
| average | | | 7.8 | 0.132 | 0.392 | 1.30 | 2.10 | 7.8 | 0.120 | 0.270 | 1.04 | 1.64 | 8.5 | 0.098 | 0.296 | 0.82 | 1.69 |
| 100 | 40 | 3 | 0 | 0.479 | 0.794 | 25.09 | 32.19 | 3 | 0.255 | 0.377 | 16.85 | 28.36 | 6 | 0.154 | 0.631 | 9.90 | 25.87 |
| 100 | 40 | 4 | 3 | 0.401 | 1.575 | 22.68 | 34.66 | 3 | 0.259 | 0.380 | 17.34 | 25.62 | 8 | 0.091 | 0.625 | 4.90 | 24.54 |
| 100 | 40 | 5 | 4 | 0.323 | 1.181 | 17.60 | 33.06 | 6 | 0.215 | 0.714 | 10.35 | 30.18 | 9 | 0.033 | 0.326 | 2.25 | 22.47 |
| 100 | 40 | 10 | 2 | 0.543 | 1.460 | 23.69 | 35.89 | 7 | 0.103 | 0.352 | 8.59 | 29.35 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 40 | 5 | 10 | 0.000 | 0.000 | 0.04 | 0.05 | 10 | 0.000 | 0.000 | 0.78 | 2.24 | 9 | 0.016 | 0.164 | 22.83 | 206.73 |
| 200 | 40 | 10 | 10 | 0.000 | 0.000 | 0.03 | 0.32 | 9 | 0.019 | 0.192 | 3.65 | 36.43 | 9 | 0.017 | 0.168 | 3.84 | 35.90 |
| 200 | 40 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 400 | 40 | 10 | 10 | 0.000 | 0.000 | 0.08 | 0.13 | 10 | 0.000 | 0.000 | 0.08 | 0.09 | 10 | 0.000 | 0.000 | 0.10 | 0.30 |
| 400 | 40 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.10 | 0.32 | 10 | 0.000 | 0.000 | 0.09 | 0.26 |
| 400 | 40 | 40 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.03 | 0.16 |
| average | | | 6.9 | 0.175 | 0.501 | 8.92 | 13.63 | 7.8 | 0.085 | 0.202 | 5.78 | 15.26 | 9.1 | 0.031 | 0.191 | 4.39 | 31.62 |
| 100 | 50 | 3 | 7 | 0.284 | 1.429 | 11.66 | 37.78 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 50 | 4 | 7 | 0.337 | 1.463 | 14.17 | 31.74 | 9 | 0.083 | 0.833 | 3.17 | 31.53 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 100 | 50 | 5 | 2 | 0.546 | 1.508 | 22.45 | 36.60 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.00 | 0.00 |
| 200 | 50 | 4 | 10 | 0.000 | 0.000 | 0.03 | 0.04 | 10 | 0.000 | 0.000 | 0.05 | 0.11 | 10 | 0.000 | 0.000 | 0.18 | 1.21 |
| 200 | 50 | 5 | 10 | 0.000 | 0.000 | 0.02 | 0.06 | 10 | 0.000 | 0.000 | 0.50 | 4.30 | 7 | 0.059 | 0.205 | 18.99 | 51.35 |
| 200 | 50 | 10 | 10 | 0.000 | 0.000 | 0.87 | 8.64 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.11 | 0.55 |
| 400 | 50 | 10 | 10 | 0.000 | 0.000 | 0.06 | 0.07 | 10 | 0.000 | 0.000 | 0.23 | 1.66 | 10 | 0.000 | 0.000 | 0.09 | 0.18 |
| 400 | 50 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.00 | 10 | 0.000 | 0.000 | 0.05 | 0.09 | 10 | 0.000 | 0.000 | 0.07 | 0.11 |
| average | | | 8.3 | 0.146 | 0.550 | 6.16 | 14.37 | 9.9 | 0.010 | 0.104 | 0.50 | 4.71 | 9.6 | 0.007 | 0.026 | 2.43 | 6.68 |
| overall average | | | 8.8 | 0.094 | 0.322 | 2.20 | 3.94 | 9.3 | 0.046 | 0.178 | 1.08 | 2.99 | 9.6 | 0.023 | 0.083 | 1.05 | 5.49 |

**Table 5. Results for Classes 10–12 ($KPP$ instances).**

| | | | $p_j \in [500, 10000]$ | | | | | $p_j \in [1000, 10000]$ | | | | | $p_j \in [1500, 10000]$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ | $opt$ | $\%g$ | $\%g_M$ | $t$ | $t_M$ |
| 24 | 8 | 3 | 10 | 0.000 | 0.000 | 0.35 | 0.60 | 10 | 0.000 | 0.000 | 0.32 | 0.60 | 10 | 0.000 | 0.000 | 0.31 | 0.60 |
| 32 | 8 | 4 | 0 | 0.071 | 0.100 | 1.23 | 1.53 | 0 | 0.056 | 0.079 | 1.21 | 1.36 | 0 | 0.063 | 0.116 | 1.19 | 1.30 |
| 40 | 8 | 5 | 0 | 0.032 | 0.053 | 1.38 | 1.56 | 0 | 0.025 | 0.035 | 1.36 | 1.53 | 0 | 0.025 | 0.037 | 1.35 | 1.50 |
| 80 | 8 | 10 | 0 | 0.004 | 0.007 | 2.98 | 3.38 | 0 | 0.003 | 0.005 | 3.04 | 3.35 | 1 | 0.003 | 0.005 | 2.66 | 3.37 |
| 120 | 8 | 15 | 2 | 0.001 | 0.001 | 5.06 | 6.62 | 5 | 0.001 | 0.001 | 2.92 | 6.47 | 2 | 0.001 | 0.001 | 4.92 | 6.32 |
| 160 | 8 | 20 | 7 | 0.000 | 0.001 | 4.08 | 9.81 | 7 | 0.000 | 0.001 | 3.24 | 9.81 | 9 | 0.000 | 0.001 | 1.45 | 8.83 |
| 200 | 8 | 25 | 10 | 0.000 | 0.000 | 1.86 | 12.71 | 8 | 0.000 | 0.001 | 3.18 | 14.69 | 8 | 0.000 | 0.001 | 2.69 | 12.34 |
| average | | | 4.1 | 0.015 | 0.023 | 2.42 | 5.17 | 4.3 | 0.012 | 0.018 | 2.18 | 5.40 | 4.3 | 0.013 | 0.023 | 2.08 | 4.90 |
| 30 | 10 | 3 | 10 | 0.000 | 0.000 | 0.73 | 1.42 | 10 | 0.000 | 0.000 | 0.68 | 1.39 | 10 | 0.000 | 0.000 | 0.72 | 1.34 |
| 40 | 10 | 4 | 0 | 0.076 | 0.127 | 1.77 | 1.97 | 0 | 0.063 | 0.095 | 1.82 | 2.05 | 0 | 0.063 | 0.081 | 1.68 | 1.95 |
| 50 | 10 | 5 | 0 | 0.034 | 0.051 | 2.37 | 2.91 | 0 | 0.026 | 0.038 | 2.28 | 2.90 | 0 | 0.024 | 0.038 | 2.17 | 2.50 |
| 100 | 10 | 10 | 0 | 0.005 | 0.006 | 5.90 | 6.99 | 0 | 0.004 | 0.006 | 5.74 | 6.73 | 0 | 0.004 | 0.006 | 5.42 | 6.71 |
| 150 | 10 | 15 | 2 | 0.001 | 0.003 | 9.78 | 13.60 | 4 | 0.001 | 0.001 | 7.22 | 11.91 | 3 | 0.001 | 0.002 | 8.10 | 13.76 |
| 200 | 10 | 20 | 3 | 0.001 | 0.001 | 12.86 | 20.12 | 8 | 0.000 | 0.001 | 3.98 | 20.02 | 8 | 0.000 | 0.001 | 4.47 | 20.23 |
| 250 | 10 | 25 | 8 | 0.000 | 0.001 | 6.45 | 31.02 | 8 | 0.000 | 0.001 | 7.95 | 25.12 | 9 | 0.000 | 0.001 | 6.03 | 29.90 |
| average | | | 3.3 | 0.017 | 0.027 | 5.69 | 11.15 | 4.3 | 0.014 | 0.020 | 4.24 | 10.02 | 4.3 | 0.013 | 0.018 | 4.08 | 10.91 |
| 39 | 13 | 3 | 2 | 0.566 | 1.698 | 2.06 | 2.65 | 2 | 0.515 | 1.539 | 2.06 | 2.74 | 2 | 0.469 | 1.394 | 2.11 | 2.67 |
| 52 | 13 | 4 | 0 | 0.080 | 0.111 | 3.22 | 3.61 | 0 | 0.062 | 0.087 | 3.06 | 3.45 | 0 | 0.062 | 0.087 | 3.10 | 3.49 |
| 65 | 13 | 5 | 0 | 0.032 | 0.053 | 4.46 | 4.96 | 1 | 0.022 | 0.042 | 4.00 | 4.73 | 1 | 0.016 | 0.033 | 3.75 | 4.70 |
| 130 | 13 | 10 | 1 | 0.003 | 0.006 | 11.70 | 15.20 | 3 | 0.003 | 0.006 | 10.12 | 17.28 | 3 | 0.002 | 0.004 | 9.02 | 14.11 |
| 195 | 13 | 15 | 5 | 0.001 | 0.003 | 11.05 | 25.37 | 5 | 0.001 | 0.001 | 12.93 | 27.90 | 7 | 0.001 | 0.002 | 6.86 | 26.72 |
| 260 | 13 | 20 | 5 | 0.000 | 0.001 | 21.90 | 46.38 | 8 | 0.000 | 0.001 | 8.56 | 41.63 | 7 | 0.000 | 0.001 | 12.74 | 44.61 |
| 325 | 13 | 25 | 10 | 0.000 | 0.000 | 4.98 | 21.78 | 8 | 0.000 | 0.001 | 13.41 | 64.62 | 9 | 0.000 | 0.001 | 6.51 | 57.53 |
| average | | | 3.3 | 0.098 | 0.267 | 8.48 | 17.13 | 3.9 | 0.086 | 0.239 | 7.73 | 23.19 | 4.1 | 0.079 | 0.217 | 6.30 | 21.98 |
| 45 | 15 | 3 | 2 | 0.208 | 0.526 | 2.67 | 3.81 | 2 | 0.190 | 0.475 | 2.70 | 3.84 | 2 | 0.169 | 0.435 | 2.57 | 3.75 |
| 60 | 15 | 4 | 0 | 0.070 | 0.097 | 4.55 | 5.49 | 0 | 0.065 | 0.083 | 4.70 | 5.13 | 0 | 0.057 | 0.083 | 4.49 | 4.98 |
| 75 | 15 | 5 | 2 | 0.023 | 0.043 | 4.80 | 6.77 | 2 | 0.023 | 0.044 | 5.09 | 6.99 | 1 | 0.019 | 0.035 | 5.19 | 6.23 |
| 150 | 15 | 10 | 3 | 0.003 | 0.006 | 12.50 | 18.95 | 4 | 0.003 | 0.006 | 11.48 | 21.30 | 4 | 0.003 | 0.005 | 10.79 | 21.02 |
| 225 | 15 | 15 | 8 | 0.000 | 0.001 | 8.29 | 40.95 | 6 | 0.001 | 0.002 | 13.26 | 37.53 | 8 | 0.000 | 0.001 | 7.39 | 37.36 |
| 300 | 15 | 20 | 7 | 0.000 | 0.001 | 18.24 | 63.17 | 8 | 0.000 | 0.001 | 16.57 | 56.16 | 7 | 0.000 | 0.001 | 22.26 | 61.41 |
| 375 | 15 | 25 | 8 | 0.000 | 0.001 | 25.81 | 96.75 | 9 | 0.000 | 0.001 | 19.81 | 128.31 | 7 | 0.000 | 0.001 | 32.30 | 125.63 |
| average | | | 4.3 | 0.044 | 0.096 | 10.98 | 33.70 | 4.4 | 0.040 | 0.087 | 10.52 | 37.04 | 4.1 | 0.035 | 0.080 | 12.14 | 37.20 |
| 54 | 18 | 3 | 0 | 0.245 | 0.367 | 4.73 | 6.08 | 0 | 0.207 | 0.333 | 5.02 | 5.79 | 0 | 0.196 | 0.302 | 5.09 | 6.17 |
| 72 | 18 | 4 | 0 | 0.070 | 0.096 | 7.86 | 9.89 | 0 | 0.063 | 0.086 | 7.42 | 8.85 | 0 | 0.060 | 0.082 | 7.24 | 9.11 |
| 90 | 18 | 5 | 1 | 0.020 | 0.039 | 9.18 | 12.00 | 0 | 0.019 | 0.035 | 9.96 | 11.97 | 1 | 0.018 | 0.036 | 9.06 | 12.06 |
| 180 | 18 | 10 | 2 | 0.003 | 0.006 | 22.51 | 34.70 | 8 | 0.001 | 0.004 | 6.97 | 34.71 | 7 | 0.001 | 0.003 | 10.05 | 33.12 |
| 270 | 18 | 15 | 10 | 0.000 | 0.000 | 1.01 | 1.60 | 5 | 0.001 | 0.001 | 28.64 | 63.28 | 7 | 0.000 | 0.001 | 20.61 | 73.49 |
| 360 | 18 | 20 | 9 | 0.000 | 0.001 | 12.34 | 112.36 | 8 | 0.000 | 0.001 | 23.25 | 115.74 | 9 | 0.000 | 0.001 | 12.10 | 110.95 |
| average | | | 3.7 | 0.056 | 0.085 | 9.61 | 29.44 | 3.5 | 0.048 | 0.077 | 13.54 | 40.06 | 4.0 | 0.046 | 0.071 | 10.69 | 40.82 |
| 60 | 20 | 3 | 0 | 0.213 | 0.277 | 6.63 | 7.39 | 0 | 0.193 | 0.262 | 6.65 | 7.36 | 0 | 0.173 | 0.226 | 6.59 | 7.28 |
| 80 | 20 | 4 | 0 | 0.060 | 0.086 | 9.80 | 12.40 | 0 | 0.058 | 0.069 | 9.75 | 11.23 | 0 | 0.049 | 0.078 | 9.80 | 13.25 |
| 100 | 20 | 5 | 2 | 0.017 | 0.043 | 11.40 | 16.13 | 5 | 0.011 | 0.039 | 7.60 | 14.98 | 3 | 0.011 | 0.036 | 9.01 | 15.32 |
| 200 | 20 | 10 | 3 | 0.003 | 0.006 | 30.94 | 52.35 | 4 | 0.002 | 0.006 | 25.35 | 57.29 | 8 | 0.001 | 0.004 | 9.43 | 46.82 |
| 300 | 20 | 15 | 8 | 0.000 | 0.001 | 19.18 | 99.00 | 8 | 0.000 | 0.001 | 17.51 | 86.76 | 10 | 0.000 | 0.000 | 1.34 | 2.25 |
| 400 | 20 | 20 | 10 | 0.000 | 0.000 | 1.76 | 2.58 | 7 | 0.000 | 0.001 | 48.84 | 164.88 | 10 | 0.000 | 0.000 | 1.32 | 2.07 |
| average | | | 3.8 | 0.049 | 0.069 | 13.29 | 31.64 | 4.0 | 0.044 | 0.063 | 19.28 | 57.08 | 5.2 | 0.039 | 0.057 | 6.25 | 14.50 |
| 75 | 25 | 3 | 0 | 0.177 | 0.231 | 11.50 | 14.18 | 0 | 0.158 | 0.208 | 11.41 | 14.06 | 0 | 0.151 | 0.170 | 11.30 | 13.65 |
| 100 | 25 | 4 | 1 | 0.040 | 0.072 | 15.79 | 23.74 | 0 | 0.038 | 0.073 | 16.02 | 25.71 | 0 | 0.028 | 0.052 | 14.71 | 21.94 |
| 125 | 25 | 5 | 3 | 0.011 | 0.033 | 18.21 | 28.65 | 2 | 0.013 | 0.033 | 21.07 | 28.20 | 6 | 0.006 | 0.030 | 12.77 | 28.03 |
| 250 | 25 | 10 | 8 | 0.000 | 0.002 | 17.50 | 86.61 | 9 | 0.001 | 0.005 | 9.42 | 76.18 | 8 | 0.000 | 0.002 | 16.98 | 91.37 |
| 375 | 25 | 15 | 9 | 0.000 | 0.001 | 20.96 | 189.89 | 9 | 0.000 | 0.001 | 18.52 | 165.76 | 10 | 0.000 | 0.000 | 1.89 | 2.79 |
| average | | | 4.2 | 0.046 | 0.068 | 16.79 | 68.61 | 4.0 | 0.042 | 0.064 | 15.29 | 61.98 | 4.8 | 0.037 | 0.051 | 11.53 | 31.56 |
| 90 | 30 | 3 | 0 | 0.169 | 0.196 | 19.32 | 22.02 | 0 | 0.156 | 0.187 | 19.41 | 22.02 | 0 | 0.137 | 0.174 | 19.03 | 21.71 |
| 120 | 30 | 4 | 0 | 0.027 | 0.050 | 25.06 | 29.65 | 0 | 0.028 | 0.055 | 23.56 | 27.98 | 2 | 0.024 | 0.054 | 20.90 | 29.07 |
| 150 | 30 | 5 | 6 | 0.005 | 0.023 | 20.33 | 47.16 | 6 | 0.004 | 0.019 | 18.89 | 55.34 | 8 | 0.004 | 0.025 | 13.50 | 55.06 |
| 300 | 30 | 10 | 9 | 0.000 | 0.004 | 18.62 | 152.93 | 10 | 0.000 | 0.000 | 1.56 | 2.60 | 9 | 0.000 | 0.002 | 15.86 | 141.61 |
| average | | | 3.8 | 0.050 | 0.068 | 20.83 | 62.94 | 4.0 | 0.047 | 0.065 | 15.86 | 26.99 | 4.8 | 0.041 | 0.064 | 17.32 | 61.86 |
| overall average | | | 3.8 | 0.046 | 0.090 | 10.16 | 29.21 | 4.1 | 0.041 | 0.081 | 10.40 | 31.23 | 4.4 | 0.038 | 0.074 | 8.18 | 25.76 |

**Table 6. Results for Classes 13–15 ($KPP$ perfect packing instances).**

| | | | $z = 1000$ | | | | | $z = 5000$ | | | | | $z = 10000$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | *opt* | %$g$ | %$g_M$ | $t$ | $t_M$ | *opt* | %$g$ | %$g_M$ | $t$ | $t_M$ | *opt* | %$g$ | %$g_M$ | $t$ | $t_M$ |
| 24 | 8 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.02 | 0.04 |
| 32 | 8 | 4 | 9 | 0.010 | 0.100 | 0.21 | 1.25 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.03 | 0.07 |
| 40 | 8 | 5 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 0 | 0.036 | 0.060 | 1.48 | 1.65 | 1 | 0.036 | 0.060 | 1.32 | 1.59 |
| 80 | 8 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.05 | 0.29 | 4 | 0.006 | 0.010 | 1.71 | 2.86 |
| 120 | 8 | 15 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.02 | 0.04 |
| 160 | 8 | 20 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.02 | 0.04 |
| 200 | 8 | 25 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.02 | 0.03 |
| average | | | 9.9 | 0.001 | 0.014 | 0.03 | 0.19 | 8.6 | 0.005 | 0.009 | 0.23 | 0.29 | 7.9 | 0.006 | 0.010 | 0.45 | 0.67 |
| 30 | 10 | 3 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.03 | 0.07 |
| 40 | 10 | 4 | 2 | 0.080 | 0.100 | 1.44 | 2.06 | 5 | 0.044 | 0.120 | 0.89 | 1.88 | 8 | 0.021 | 0.110 | 0.32 | 1.63 |
| 50 | 10 | 5 | 10 | 0.000 | 0.000 | 0.02 | 0.08 | 2 | 0.030 | 0.060 | 1.88 | 2.76 | 0 | 0.042 | 0.060 | 2.42 | 2.67 |
| 100 | 10 | 10 | 10 | 0.000 | 0.000 | 0.00 | 0.01 | 10 | 0.000 | 0.000 | 0.04 | 0.12 | 6 | 0.004 | 0.010 | 1.75 | 4.56 |
| 150 | 10 | 15 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.02 | 0.04 | 10 | 0.000 | 0.000 | 0.08 | 0.30 |
| 200 | 10 | 20 | 10 | 0.000 | 0.000 | 0.01 | 0.01 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.06 |
| 250 | 10 | 25 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.02 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.04 |
| average | | | 8.9 | 0.011 | 0.014 | 0.21 | 0.31 | 8.1 | 0.011 | 0.026 | 0.41 | 0.70 | 7.7 | 0.010 | 0.026 | 0.67 | 1.33 |
| 39 | 13 | 3 | 10 | 0.000 | 0.000 | 0.01 | 0.04 | 10 | 0.000 | 0.000 | 0.02 | 0.04 | 10 | 0.000 | 0.000 | 0.04 | 0.08 |
| 52 | 13 | 4 | 2 | 0.080 | 0.100 | 2.40 | 3.24 | 0 | 0.092 | 0.120 | 3.24 | 3.50 | 2 | 0.074 | 0.120 | 2.73 | 3.65 |
| 65 | 13 | 5 | 10 | 0.000 | 0.000 | 0.06 | 0.11 | 3 | 0.016 | 0.040 | 3.43 | 5.61 | 0 | 0.033 | 0.050 | 4.71 | 5.66 |
| 130 | 13 | 10 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.06 | 0.20 | 6 | 0.004 | 0.010 | 3.37 | 8.40 |
| 195 | 13 | 15 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.04 | 0.12 | 10 | 0.000 | 0.000 | 0.05 | 0.07 |
| 260 | 13 | 20 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.04 | 10 | 0.000 | 0.000 | 0.05 | 0.07 |
| 325 | 13 | 25 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.04 | 10 | 0.000 | 0.000 | 0.05 | 0.08 |
| average | | | 8.9 | 0.011 | 0.014 | 0.36 | 0.50 | 7.6 | 0.015 | 0.023 | 0.98 | 1.36 | 6.9 | 0.016 | 0.026 | 1.57 | 2.57 |
| 45 | 15 | 3 | 9 | 0.030 | 0.300 | 0.31 | 2.99 | 10 | 0.000 | 0.000 | 0.03 | 0.04 | 10 | 0.000 | 0.000 | 0.05 | 0.10 |
| 60 | 15 | 4 | 3 | 0.070 | 0.100 | 2.85 | 4.70 | 0 | 0.082 | 0.120 | 4.66 | 5.18 | 1 | 0.081 | 0.120 | 4.36 | 5.23 |
| 75 | 15 | 5 | 10 | 0.000 | 0.000 | 0.09 | 0.38 | 4 | 0.012 | 0.020 | 4.49 | 9.35 | 0 | 0.027 | 0.040 | 7.35 | 9.67 |
| 150 | 15 | 10 | 10 | 0.000 | 0.000 | 0.01 | 0.03 | 10 | 0.000 | 0.000 | 0.06 | 0.14 | 10 | 0.000 | 0.000 | 0.21 | 1.31 |
| 225 | 15 | 15 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.04 | 10 | 0.000 | 0.000 | 0.06 | 0.11 |
| 300 | 15 | 20 | 10 | 0.000 | 0.000 | 0.02 | 0.02 | 10 | 0.000 | 0.000 | 0.04 | 0.04 | 10 | 0.000 | 0.000 | 0.07 | 0.10 |
| 375 | 15 | 25 | 10 | 0.000 | 0.000 | 0.01 | 0.02 | 10 | 0.000 | 0.000 | 0.03 | 0.05 | 10 | 0.000 | 0.000 | 0.05 | 0.10 |
| average | | | 8.9 | 0.014 | 0.057 | 0.47 | 1.17 | 7.7 | 0.013 | 0.020 | 1.33 | 2.12 | 7.3 | 0.015 | 0.023 | 1.74 | 2.37 |
| 54 | 18 | 3 | 9 | 0.020 | 0.200 | 0.52 | 4.33 | 10 | 0.000 | 0.000 | 0.04 | 0.07 | 10 | 0.000 | 0.000 | 0.08 | 0.13 |
| 72 | 18 | 4 | 5 | 0.050 | 0.100 | 3.43 | 7.36 | 0 | 0.060 | 0.120 | 7.71 | 8.96 | 0 | 0.098 | 0.130 | 8.02 | 9.66 |
| 90 | 18 | 5 | 10 | 0.000 | 0.000 | 0.22 | 0.73 | 4 | 0.014 | 0.040 | 8.31 | 16.49 | 0 | 0.027 | 0.040 | 12.76 | 15.75 |
| 180 | 18 | 10 | 10 | 0.000 | 0.000 | 0.05 | 0.21 | 10 | 0.000 | 0.000 | 0.23 | 1.90 | 9 | 0.001 | 0.010 | 2.33 | 20.78 |
| 270 | 18 | 15 | 10 | 0.000 | 0.000 | 0.03 | 0.06 | 10 | 0.000 | 0.000 | 0.06 | 0.15 | 10 | 0.000 | 0.000 | 0.10 | 0.26 |
| 360 | 18 | 20 | 10 | 0.000 | 0.000 | 0.03 | 0.03 | 10 | 0.000 | 0.000 | 0.06 | 0.10 | 10 | 0.000 | 0.000 | 0.10 | 0.27 |
| average | | | 9.0 | 0.012 | 0.050 | 0.71 | 2.12 | 7.3 | 0.012 | 0.027 | 2.74 | 4.61 | 6.5 | 0.021 | 0.030 | 3.90 | 7.81 |
| 60 | 20 | 3 | 6 | 0.090 | 0.300 | 2.66 | 5.54 | 9 | 0.024 | 0.240 | 0.60 | 5.32 | 10 | 0.000 | 0.000 | 0.11 | 0.22 |
| 80 | 20 | 4 | 7 | 0.030 | 0.100 | 3.49 | 10.82 | 0 | 0.068 | 0.100 | 10.55 | 12.38 | 0 | 0.065 | 0.100 | 11.18 | 14.28 |
| 100 | 20 | 5 | 10 | 0.000 | 0.000 | 3.76 | 33.20 | 5 | 0.010 | 0.020 | 12.38 | 26.07 | 1 | 0.027 | 0.050 | 19.96 | 34.83 |
| 200 | 20 | 10 | 10 | 0.000 | 0.000 | 0.02 | 0.05 | 10 | 0.000 | 0.000 | 0.06 | 0.14 | 10 | 0.000 | 0.000 | 0.21 | 0.49 |
| 300 | 20 | 15 | 10 | 0.000 | 0.000 | 0.04 | 0.12 | 10 | 0.000 | 0.000 | 0.07 | 0.14 | 10 | 0.000 | 0.000 | 0.10 | 0.15 |
| 400 | 20 | 20 | 10 | 0.000 | 0.000 | 0.06 | 0.25 | 10 | 0.000 | 0.000 | 0.06 | 0.16 | 10 | 0.000 | 0.000 | 0.12 | 0.24 |
| average | | | 8.8 | 0.020 | 0.067 | 1.67 | 8.33 | 7.3 | 0.017 | 0.060 | 3.95 | 7.37 | 6.8 | 0.015 | 0.025 | 5.28 | 8.37 |
| 75 | 25 | 3 | 2 | 0.160 | 0.200 | 7.62 | 9.39 | 10 | 0.000 | 0.000 | 0.09 | 0.17 | 10 | 0.000 | 0.000 | 0.15 | 0.23 |
| 100 | 25 | 4 | 8 | 0.020 | 0.100 | 7.15 | 18.96 | 0 | 0.060 | 0.100 | 19.08 | 23.51 | 0 | 0.064 | 0.100 | 19.97 | 24.82 |
| 125 | 25 | 5 | 10 | 0.000 | 0.000 | 1.38 | 2.93 | 10 | 0.000 | 0.000 | 7.97 | 26.50 | 4 | 0.008 | 0.020 | 36.75 | 72.70 |
| 250 | 25 | 10 | 10 | 0.000 | 0.000 | 0.15 | 0.47 | 10 | 0.000 | 0.000 | 2.68 | 24.79 | 10 | 0.000 | 0.000 | 0.31 | 0.67 |
| 375 | 25 | 15 | 10 | 0.000 | 0.000 | 0.16 | 0.72 | 10 | 0.000 | 0.000 | 0.09 | 0.13 | 10 | 0.000 | 0.000 | 0.19 | 0.59 |
| average | | | 8.0 | 0.036 | 0.060 | 3.29 | 6.49 | 8.0 | 0.012 | 0.020 | 5.98 | 15.02 | 6.8 | 0.014 | 0.024 | 11.47 | 19.80 |
| 90 | 30 | 3 | 0 | 0.180 | 0.200 | 13.65 | 14.48 | 1 | 0.148 | 0.200 | 13.43 | 15.89 | 6 | 0.059 | 0.190 | 5.86 | 15.84 |
| 120 | 30 | 4 | 4 | 0.060 | 0.100 | 23.07 | 42.53 | 0 | 0.054 | 0.100 | 37.67 | 44.47 | 0 | 0.069 | 0.090 | 39.86 | 48.83 |
| 150 | 30 | 5 | 10 | 0.000 | 0.000 | 1.99 | 6.65 | 10 | 0.000 | 0.000 | 11.95 | 42.28 | 2 | 0.011 | 0.020 | 78.27 | 152.07 |
| 300 | 30 | 10 | 10 | 0.000 | 0.000 | 0.09 | 0.37 | 10 | 0.000 | 0.000 | 0.78 | 3.83 | 10 | 0.000 | 0.000 | 1.37 | 8.73 |
| average | | | 6.0 | 0.060 | 0.075 | 9.70 | 16.01 | 5.3 | 0.050 | 0.075 | 15.95 | 26.62 | 4.5 | 0.035 | 0.075 | 31.34 | 56.37 |
| overall average | | | 8.7 | 0.018 | 0.041 | 1.57 | 3.56 | 7.6 | 0.015 | 0.030 | 3.15 | 5.81 | 6.9 | 0.015 | 0.027 | 5.48 | 9.60 |

**Table 7. Results for real world PCB instances.**

| | | | | | | | Percentage error | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n$ | $m$ | $k$ | $L$ | $z$ | *opt* | $LPT_k$ | $MS_k$ | $MS2_k$ | *Scatter 0* | *Scatter 1* | *B&B* | time |
| $PCB_1$ | 403 | 5 | 81 | 115340 | 115341 | 0 | 0.0139 | 16.3092 | 0.1621 | 0.0009 | 0.0009 | 0.0009 | 5.46 |
| | 403 | 10 | 41 | 57670 | 57672 | 0 | 0.1075 | 15.5419 | 9.7312 | 0.0035 | 0.0035 | 0.0035 | 10.36 |
| | 403 | 20 | 21 | 32274 | 32274 | 1 | 0.0000 | 10.7920 | 10.8106 | 0.0000 | 0.0000 | 0.0000 | 0.01 |
| | 1341 | 5 | 269 | 512403 | 512403 | 1 | 0.0135 | 13.0760 | 12.4379 | 0.0000 | 0.0000 | 0.0000 | 1.13 |
| | 1341 | 10 | 135 | 256202 | 256203 | 0 | 0.0297 | 12.9757 | 6.2041 | 0.0004 | 0.0004 | 0.0004 | 67.55 |
| | 1341 | 20 | 68 | 128101 | 128103 | 0 | 0.0390 | 12.3434 | 10.7954 | 0.0016 | 0.0016 | 0.0016 | 130.62 |
| | 1417 | 5 | 284 | 467925 | 467925 | 1 | 0.0077 | 14.4190 | 13.8144 | 0.0000 | 0.0000 | 0.0000 | 0.64 |
| | 1417 | 10 | 142 | 233963 | 233964 | 0 | 0.0043 | 14.3809 | 11.1663 | 0.0004 | 0.0004 | 0.0004 | 69.62 |
| | 1417 | 20 | 71 | 116982 | 116982 | 1 | 0.1359 | 14.4868 | 14.4407 | 0.0000 | 0.0000 | 0.0000 | 2.79 |
| | 1312 | 5 | 263 | 446266 | 446268 | 0 | 0.0139 | 14.0739 | 1.6817 | 0.0004 | 0.0004 | 0.0004 | 34.95 |
| | 1312 | 10 | 132 | 223133 | 223134 | 0 | 0.0341 | 13.1778 | 4.0191 | 0.0004 | 0.0004 | 0.0004 | 63.23 |
| | 1312 | 20 | 66 | 111567 | 111567 | 1 | 0.0215 | 14.4855 | 14.4371 | 0.0000 | 0.0000 | 0.0000 | 0.49 |
| | average | | | | | | 0.0351 | 13.8385 | 9.1417 | 0.0006 | 0.0006 | 0.0006 | 32.24 |
| $PCB_2$ | 314 | 5 | 63 | 114324 | 114324 | 1 | 0.0184 | 12.4541 | 11.3546 | 0.0000 | 0.0000 | 0.0000 | 0.05 |
| | 314 | 10 | 32 | 57162 | 57162 | 1 | 0.0052 | 12.0710 | 11.4884 | 0.0000 | 0.0000 | 0.0000 | 0.01 |
| | 314 | 20 | 16 | 31965 | 31965 | 1 | 0.0000 | 8.9911 | 8.6532 | 0.0000 | 0.0000 | 0.0000 | 0.01 |
| | 972 | 5 | 195 | 512007 | 512007 | 1 | 0.0129 | 11.8006 | 11.1432 | 0.0000 | 0.0000 | 0.0000 | 1.36 |
| | 972 | 10 | 98 | 256004 | 256005 | 0 | 0.0238 | 10.8190 | 8.9522 | 0.0004 | 0.0004 | 0.0004 | 45.17 |
| | 972 | 20 | 49 | 128002 | 128004 | 0 | 0.0156 | 11.0287 | 9.0881 | 0.0016 | 0.0016 | 0.0016 | 84.79 |
| | 1084 | 5 | 217 | 459495 | 459495 | 1 | 0.0104 | 13.1198 | 12.5747 | 0.0000 | 0.0000 | 0.0000 | 1.00 |
| | 1084 | 10 | 109 | 229749 | 229749 | 1 | 0.0261 | 13.0264 | 10.7047 | 0.0000 | 0.0000 | 0.0000 | 6.23 |
| | 1084 | 20 | 55 | 114876 | 114876 | 1 | 0.0470 | 12.6893 | 10.8378 | 0.0000 | 0.0000 | 0.0000 | 9.35 |
| | 1056 | 5 | 212 | 445266 | 445266 | 1 | 0.0168 | 13.0520 | 12.4375 | 0.0000 | 0.0000 | 0.0000 | 1.44 |
| | 1056 | 10 | 106 | 222633 | 222633 | 1 | 0.0364 | 13.1625 | 13.0479 | 0.0000 | 0.0000 | 0.0000 | 1.55 |
| | 1056 | 20 | 53 | 111317 | 111318 | 0 | 0.0252 | 13.0825 | 11.3253 | 0.0009 | 0.0009 | 0.0009 | 109.87 |
| | average | | | | | | 0.0198 | 12.1081 | 10.9673 | 0.0002 | 0.0002 | 0.0002 | 21.74 |
| | overall average | | | | | | 0.0275 | 12.9733 | 10.0545 | 0.0004 | 0.0004 | 0.0004 | 26.99 |