Ann Oper Res (2010) 181: 443–462
DOI 10.1007/s10479-010-0757-3

Solving the Rectangular assignment problem and applications

J. Bijsterbosch · A. Volgenant

Published online: 5 June 2010

Abstract The rectangular assignment problem is a generalization of the linear assignment problem (LAP): one wants to assign a number of persons to a smaller number of jobs, minimizing the total corresponding costs. Applications are, e.g., in the fields of object recognition and scheduling. Further, we show how it can be used to solve variants of the LAP, such as the k-cardinality LAP and the LAP with outsourcing by transformation. We introduce a transformation to solve the machine replacement LAP.

We describe improvements of a LAP-algorithm for the rectangular problem, in general and slightly adapted for these variants, based on the structure of the corresponding cost matrices. For these problem instances, we compared the best special codes from literature to our codes, which are more general and easier to implement. The improvements lead to more efficient and robust codes, making them competitive on all problem instances and often showing much shorter computing times.

Keywords Linear assignment · k-Cardinality linear assignment · Linear assignment problem with outsourcing · Machine replacement

1 Introduction

The *linear assignment problem* (LAP), also denoted as the *weighted bipartite matching problem*, is well known and intensively studied. It is defined on two equally sized node sets N_1 and N_2, and a set of feasible assignments $\mathcal{A} = \{(i, j)|i \in N_1,\ j \in N_2\}$, where each arc $(i, j) \in \mathcal{A}$ has a cost c_{ij}. The objective is to assign each node in N_1 to exactly one node in N_2, such that the total cost is minimized. For convenience we refer to nodes in N_1 as *persons* and in N_2 as *jobs*; we denote $|N_1|$ as n_1 and $|N_2|$ as n_2. Allowing $n_1 \neq n_2$, say $n_1 < n_2$, is a generalization of the LAP known as the *rectangular* LAP (RLAP). With the binary

J. Bijsterbosch · A. Volgenant (✉)
Operations Research Group, Faculty of Economics and Econometrics, University of Amsterdam, Roetersstraat 11, 1018 WB Amsterdam, The Netherlands
e-mail: a.volgenant@uva.nl

variables: $x_{ij} = 1$, if person i is assigned to job j and 0 otherwise, and the corresponding costs c_{ij} the RLAP can be stated as

$$\begin{array}{lll} \text{(RLAP)} & \min & \sum_{i\in N_1}\sum_{j\in N_2} c_{ij}x_{ij} \\ & \text{subject to} & \sum_{j\in N_2} x_{ij} = 1, \quad i \in N_1 \\ & & \sum_{i\in N_1} x_{ij} \le 1, \quad j \in N_2 \\ & & x_{ij} \in \{0,1\}, \quad (i,j) \in \mathcal{A} \end{array}$$

and the corresponding dual problem with the *dual variables* or *node potentials* u_i and v_j as:

$$\begin{array}{lll} \text{(DRLAP)} & \max & \sum_{i\in N_1} u_i + \sum_{j\in N_2} v_j \\ & \text{subject to} & c_{ij} - u_i - v_j \ge 0, \quad (i,j) \in \mathcal{A} \\ & & v_j \le 0, \quad j \in N_2 \end{array}$$

To solve the RLAP by extending it to a (square) LAP by adding dummy variables is well known and goes back to Egervary and Konig, see, e.g., Burkard et al. (2009) for further reading on this issue. The transformation increases the computational time by increasing the size of the problem and introduces degeneration into it, i. e., each optimal solution of the original problem instance corresponds generally to $(n_2 - n_1)!$ optimal solutions of the extended instance. Hence LAP algorithms have been modified for rectangularity—see, e. g., Bertsekas et al. (1993) or Volgenant (1996).

(R)LAP is a relaxation of problems as the traveling salesman, the quadratic assignment and vehicle routing. Application fields are, e.g., 1. *scheduling* and 2. *object recognition*.

1. *Scheduling*: Mosheiov and Yovel (2006) formulates the single machine problem with unit processing times as an RLAP that minimizes the sum of the earliness and tardiness costs.

   Pinedo (2002) considers parallel scheduling to minimize the sum of the flow times of all jobs, e.g., programs on computer processors. Let n jobs to be processed on one of m machines, with p_{ij} the time to process job i on machine j. The flow time of all the jobs assigned to machine j is found by multiplying the processing times, of the last job by 1, of the second last job by 2, etc., and then by summing these values. One can assign a job i in $n \times m$ ways, to one of the machines and to one of the kth to last positions on the machine, with cost $c_{ij} = kp_{ij}$. Thus the problem is an RLAP of size $n \times nm$.
2. *Object recognition*: We mention two applications:

   Handwriting recognition: In (Chinese) handwriting, each character can be viewed as a set of, line segments to be used in a similarity measure between an unknown input character and a reference character, to find an optimal matching among their sets of line segments, say Γ_1 and Γ_2. By comparing the input character to all reference characters, it is recognized as the one with the highest similarity (or lowest cost) if this similarity exceeds a given threshold, and rejected otherwise. The problem to find an optimal match reduces to a LAP, see Hsieh et al. (1995), with the cost of assigning line segments $i \in \Gamma_1$ and $j \in \Gamma_2$, depending, e.g., on their lengths and their angles. The problem is an RLAP if $|\Gamma_1| \ne |\Gamma_2|$.

   Multi-object tracking in air traffic control or radar tracking, links targets with tracks. One can partition sensor observations into tracks in some optimal way, to accurately estimate the real tracks. Two sensors located at different sites provide each a line an object must lie on; the intersection of the two lines determines the location of the object in 3-dimensional space. To locate the objects in time, each sensor provides a set of lines.

Taking measurements at distinct times, the new sensor measurements, the targets, must be matched with the predicted positions of the existing tracks. Due to false alarms, missed detections and inaccuracy some measurements cannot be matched to (say n_2) targets, see Bertsekas and Castañon (1992). With say $n_1 (< n_2)$ remaining measurements, one can solve the problem as an RLAP of size $n_1 \times n_2$.

The following LAP variations can be solved by transforming them to an RLAP:

- k-LAP, i.e., the k-cardinality LAP, see Dell'Amico and Martello (1997), defined for a given positive integer k ($k \leq n$), is the problem to assign only k (out of n) persons to k (out of n) jobs, minimizing the total cost.
- LAPout, i.e., the LAP with outsourcing, see Wiel Vander and Sahinidis (1997), a problem with the extra possibility that each 'internal' person can either be assigned to an available 'internal' job or outsourced to an 'external' job. Similarly, each internal job can either be assigned to an available internal person or outsourced to an external person.

We introduce a transformation to solve

- k-LAPrep, i.e., the problem of replacing (at most) k machines, see Caseau and Laburthe (2000). Suppose the manager of a plant has the funds to replace k of the available machines by newer ones; then k-LAPrep is the problem to optimally replace k machines as well as to assign tasks to machines under minimal production cost.

Volgenant (1996) has adapted LAPJV to solve the RLAP; we denote this algorithm as RJV: rectangular LAPJV. It is based on the easy to implement algorithm LAPJV, see Jonker and Volgenant (1987). According to Dell'Amico and Toth (2000) or Burkard et al. (2009) it is one of the fastest to solve dense LAP instances. We study the RLAP exploiting the properties of these applications to develop codes that perform well. We describe several implementations and compare their results to (special) known codes for these applications.

Section 2 focuses on how to improve the RJV algorithm and gives computational results, just as in the Sects 3, 4 and 5, which discuss adapted RJV codes for k-LAP, LAPout and k-LAPrep. Finally, we draw conclusions.

2 Implementation improvements

We treat and discuss improvements of RJV, the *rectangular* variant of LAPJV. Computational results are given to show their efficiency.

We shortly describe the four parts of LAPJV and how they are adapted for RJV.

1. The first part of LAPJV, standard *reduction of the columns* of the cost matrix, is deleted in RJV, as no longer each job has to be done and as a consequence this reduction doesn't lead to an equivalent problem instance.
2. The *reduction transfer* part of RJV is void, as it is only valid for assigned persons and all the persons are still free.
3. The *augmenting row reduction*, the last part of the initialization, attempts to find additional assignments. This part can be applied to RJV as long as the node potentials v are initially 0, because after terminating RJV the v-values are non positive and the v-values of the free jobs are still 0. Thus RLAP has the following *optimality conditions* (the reduced costs defined as $w_{ij} = c_{ij} - u_i - v_j$): An assignment X is optimal, if there exist node potentials (u, v) for all $(i, j) \in \mathcal{A}$ with

$$w_{ij} = 0, \quad (i, j) \in X, \qquad w_{ij} \geq 0, \quad (i, j) \in \mathcal{A}$$

$$v_j = 0, \quad j \in \{k | k \in N_2, (i,k) \notin X\}, \qquad v_j \leq 0, \quad j \in N_2$$

4. The *augmentation phase* completes the often incomplete solution obtained so far by finding alternating shortest paths from free persons to free jobs. The optimality conditions remain satisfied during this part, as the v-values of the free jobs remain 0 and the v-values of the assigned jobs never increase.

We first discuss three implementations to improve the algorithm's performance.

- *Partitioning the job sets* (see Appendix A for the code)

  We partition the set $N_2(i)$ of jobs in N_2 that can be assigned to person i into the sets $P(i)$ of assigned jobs and $Q(i)$ of free jobs. By storing $Q(i)$ as a minimum heap, its root is the job with the lowest cost when assigned to person i. It is sufficient to create the heap when a person is selected and $Q(i)$ is not yet created and to update the heap only if its root job is assigned. It turned out to be inefficient to create all heaps at the start of the algorithm and to update them if an additional job is assigned. It is needless to maintain $P(i)$ for each single person as it is the same for all persons. Therefore, an array of length n_2 stores all the jobs. The implementation of the partitioning in the augmenting row reduction part appeared to be fruitless.
- β-best implementation (see Appendix B for the code)

  The β-best implementation can bring down the number of scans in $N_2(i)$, see Goldberg and Kennnedy (1995). The two best jobs for a person i can remain the same in two successive iterations of i in the augmenting row reduction part. While $N_2(i)$ is scanned to find the lowest and second lowest net values, we can determine the β smallest net values (with $\beta \geq 3$ some integer). We can also store the largest value $b(i)$ as well as the $\beta - 1$ jobs of $N_2(i)$ that belong to the $\beta - 1$ smallest net values in a set, say $H(i)$. Consider now a subsequent iteration where the algorithm has to find the lowest and second lowest net values based on an updated set of node potentials $\underline{\boldsymbol{v}}$; then if $c_{ij} - \underline{v}_j \leq b(i)$ holds for at least two of the stored jobs $j \in H(i)$, the lowest two net values still are among the stored jobs, because the values v_j monotonously decrease during the algorithm, implying that $c_{ij} - \underline{v}_j \geq b(i)$, $j \in N_2(i) \backslash H(i)$. Thus, one can first search the smallest two net values among the stored jobs whose current net values are at most $b(i)$. The β smallest net values are only redetermined by a full scanning of $j \in N_2(i)$, if all except possibly one of the stored jobs have current net values exceeding $b(i)$. In the implementation we only compute $b(i)$ and $H(i)$ if needed, as they may change otherwise.
- *Examine the free persons in a reverse order.*

  If many persons are still free after the augmenting row reduction phase, the instance at hand can be considered as hard. Then the row minimum can occur more often at the same job for more persons. Reversing the order in which the free persons are examined may be better, as these persons may decrease more substantially the $v-$values of the specific jobs, making them less attractive for the subsequent persons.

  These three implementations appear to give significant better results, with $\beta = 6$ independent of the problem size. The next two ideas appeared to be fruitless.
- *Examine the* $n_1 cheapest jobs$. For each free person i in an RLAP we only have to examine the n_1 cheapest jobs from $N_2(i)$. The proof is straightforward and therefore omitted. It appeared that finding the n_1 cheapest jobs is too time expensive.
- *Examine assigned jobs as last ones*, and assign free jobs earlier in case of ties.

The two phases in the RJV algorithm allows us to check the "difficulty" of the problem after the augmenting row reduction part and act accordingly. The number of free persons, say f, is checked; if $f \leq 0.8 n_1$ then the instance at hand is considered as easy and solved by the

```
Function RJVI;
Begin
  for cnt:=1 to 2 do AUGROWRED_6BEST;
    {the value 2 appears to be an empirical good choice}
  If f ≤ 0.8n1 then AUGMENTATION Else AUGMENTATION_P;
  Compute optimal value Optimum;
  RJVI := Optimum
End.
```

Fig. 1 The algorithm RJVI

existing augmentation procedure, else the procedure AUGMENTATION_P, augmentation with partitioning of the job set (Appendix A) is used. In addition, in this new procedure the free persons are examined in *backward* order. The rectangular LAPJV with the described improvements (summarized in Fig. 1) is referred to as RJVI.

We will compare RJVI on the following randomly generated problem classes and a benchmark class, all with dense cost matrices and created by the random generator of Knuth (1981). The classes are:

- *Uniform random*: This class is a common one to test LAP algorithms; the entries c_{ij} of the cost matrix are integers uniformly drawn on the interval $[1, R]$, with $R \in \{10^2, 10^3, 10^6\}$.
- *Geometric*: In this class, see Dell'Amico and Toth (2000), two sets A and B, are generated, each with $n_1 + n_2$ integer values, uniformly drawn in a given interval. We set c_{ij} as the truncated Euclidean distance between two points, i.e.,

$$c_{ij} = \text{trunc}\left(\sqrt{a_{ij}^2 + b_{ij}^2}\right) + 1, \quad \text{with } a_{ij} = a_i - a_{n_1+j},\ b_{ij} = b_i - b_{n_1+j} \quad (i \in N_1,\ j \in N_2).$$

- *Machol-Wien* (1976) defined a famous benchmark class of difficult (deterministic) instances by $c_{ij} = ij + 1$ $(i \in N_1,\ j \in N_2)$.
- *Randomized Machol-Wien*: Dell'Amico and Toth (2000) introduced this variant of the previous class with c_{ij} an integer value uniformly generated in $[1, ij+1]$ $(i \in N_1,\ j \in N_2)$.

The tables to come report the average CPU times (in milliseconds), neglecting times to generate the cost matrices, since they are equal for all the algorithms. The *improvement factor* ϕ is given as the improvement of the new results relative (in %) to the old ones. The RLAP algorithms were coded in Delphi 6 and run on a personal computer (AMD 1 GHz processor with 384 MB RAM, with Windows XP as operating system). They have been tested with the sizes $n_2 \in \{1000, 1500, 2000\}$ and $n_1 \in \{0.25, 0.50, 0.75, 0.90, 0.99\}n_2$. We solved 10 random instances for each pair of (n_1, n_2), except the deterministic Machol Wien class.

We compare RJVI with RJV on *rectangular* instances and with LAPJV on *square* instances. We left out results of the naive transformation into a square LAP (with $n_2 - n_1$ dummy nodes with 0-cost for all arcs connected to the jobs of N_2). It is about 2 times ($n_1 \approx n_2$) up to about 30 times slower than RJVI, we think because reduction transfer fails on the dummy rows and more persons (n_1 is larger) have to be assigned.

The results in Table 1 show that for the uniform random and the geometric class, the running times of RJVI often do not grow with larger cost ranges in contrast to RJV. Instances with the largest range are even solved as fastest for $n_1 \leq 0.9n_2$, and always faster than the middle cost range instances. We think the improvement for the geometric instances and the

Table 1 Results for the random RLAP class; RJV and RJVI times in ms

n_2	n_1	Range $[1, 10^2]$			Range $[1, 10^3]$			Range $[1, 10^6]$		
		RJV	RJVI	ϕ in %	RJV	RJVI	ϕ in %	RJV	RJVI	ϕ in %
1000	250	5	4	20.0	5	5	0.0	5	5	0.0
uniform	500	14	13	7.1	12	12	0.0	13	12	7.7
	750	27	23	14.8	23	20	13.0	23	17	26.1
	900	39	35	10.3	40	32	20.0	46	22	52.2
	990	80	75	6.3	110	105	4.5	177	70	60.5
	1000	128[a]	103	19.5	145[a]	160	−10.3	193[a]	134	30.6
Average				*13.0*			*4.5*			*29.5*
1500	375	14	13	7.1	13	13	0.0	12	12	0.0
uniform	750	38	33	13.2	27	26	3.7	27	26	3.7
	1125	68	59	13.2	51	46	9.8	57	39	31.6
	1350	94	75	20.2	85	71	16.5	111	52	53.2
	1485	127	111	12.6	265	271	−2.3	448	205	54.2
	1500	266[a]	160	39.8	370[a]	392	−5.9	511[a]	405	20.7
Average				*17.7*			*3.6*			*27.2*
2000	500	27	25	7.4	22	22	0.0	23	22	4.3
uniform	1000	77	62	19.5	48	45	6.3	50	43	14.0
	1500	135	109	19.3	89	82	7.9	101	69	31.7
	1800	179	142	20.7	155	136	12.3	200	91	54.5
	1980	222	182	18.0	502	441	12.2	768	403	47.5
	2000	461[a]	228	50.5	761[a]	756	0.7	997[a]	922	7.5
Average				*22.6*			*6.5*			*26.6*
Average uniform instances				*17.8*			*4.9*			*27.8*
1000	250	5	5	0.0	4	4	0.0	4	4	0.0
geometric	500	13	13	0.0	14	13	7.1	17	12	29.4
	750	35	32	8.6	48	36	25.0	98	28	71.4
	900	88	84	4.5	121	99	18.2	357	85	76.2
	990	304	295	3.0	442	409	7.5	1148	392	65.9
	1000	467[a]	412	11.8	488[a]	438	10.2	1260[a]	465	63.1
Average				*4.6*			*11.3*			*51.0*
1500	375	12	13	−8.3	13	13	0.0	13	13	0.0
geometric	750	33	30	9.1	34	28	17.6	41	27	34.1
	1125	81	72	11.1	99	72	27.3	196	59	69.9
	1350	215	211	1.9	297	252	15.2	960	213	77.8
	1485	870	863	0.8	1146	1082	5.6	2636	1041	60.5
	1500	1159[a]	1084	6.5	1636[a]	1467	10.3	3009[a]	1419	52.8
Average				*3.5*			*12.7*			*49.2*
2000	500	22	23	−4.5	22	23	−4.5	23	23	0.0
geometric	1000	54	57	−5.6	67	49	26.9	68	49	27.9
	1500	145	128	11.7	175	135	22.9	414	113	72.7
	1800	377	356	5.6	529	449	15.1	1574	408	74.1
	1980	1702	1609	5.5	1906	1785	6.3	4487	1863	58.5
	2000	2483[a]	2257	9.1	3424[a]	3232	5.6	6091[a]	3084	49.4
Average				*3.6*			*12.0*			*47.1*
Average geometric instances				*3.9*			*12.0*			*49.1*

[a]Results ($n_1 = n_2 = 1000$, 1500 and 2000) obtained by LAPJV

Table 2 Results for the Machol-Wien RLAP classes; RJV and RJVI times in ms

n_2	n_1	Randomized MW			Machol Wien		
		RJV	RJVI	ϕ in %	RJV	RJVI	ϕ in %
1000	250	9	6	33.3	845	144	83.0
	500	22	15	31.8	3148	1011	67.9
	750	56	29	48.2	6379	2818	55.8
	900	106	54	49.1	8586	4467	48.0
	990	259	166	35.9	9958	5623	43.5
	1000	387[a]	238	38.5	9969[a]	5642	43.4
Average				*39.5*			*56.9*
1500	375	19	14	26.3	2883	545	81.1
	750	55	32	41.8	10490	3074	70.7
	1125	133	68	48.9	21209	8703	59.0
	1350	254	112	55.9	28530	13910	51.2
	1485	657	431	34.4	32851	17715	46.1
	1500	1013[a]	639	36.9	33192[a]	17705	46.7
Average				*40.7*			*59.1*
2000	500	36	26	27.8	6799	1167	82.8
	1000	100	60	40.0	24654	6649	73.0
	1500	246	115	53.3	49996	19242	61.5
	1800	454	193	57.5	66685	30949	53.6
	1980	1288	822	36.2	77641	39856	48.7
	2000	2135[a]	1410	34.0	80174[a]	40427	49.6
Average				*41.4*			*61.5*

[a]Results ($n_1 = n_2 = 1000$, 1500 and 2000) obtained by LAPJV

Fig. 2 Rectangular cost matrix of the $k-$LAP after the transformation

$$
\begin{array}{c|cccc:cccc}
 & \multicolumn{4}{c:}{m} & \multicolumn{4}{c}{n_1-k} & \\
\hline
 & c_{11} & c_{12} & \cdots & c_{1m} & 0 & \infty & \cdots & \infty & \\
 & c_{21} & c_{22} & \cdots & c_{2m} & \infty & 0 & \cdots & \infty & \\
 & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & n_1-k \\
 & c_{n_1-k,1} & c_{n_1-k,2} & \cdots & c_{n_1-k,m} & \infty & \infty & \cdots & 0 & \\
\hdashline
 & \vdots & \vdots & \cdots & \vdots & 0 & 0 & \cdots & 0 & \\
 & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & k \\
 & c_{n_1 1} & c_{n_1 2} & \cdots & c_{n_1 m} & 0 & 0 & \cdots & 0 &
\end{array}
$$

range $[1, 10^6]$ in particular to be due to the β-best implementation and to their structure having more likely jobs competing for a smaller number of about equally desirable persons. The algorithm RJVI is robuster and performs often much better than RJV on all tested rectangular instances. On square instances, RJVI is even up to about 50% faster on average than LAPJV with only a loss of at most 10% for the uniform random class on range $[1, 10^3]$

and $n \leq 1500$. The random Machol Wien instances (Table 2) show about the same gain up to size 2000. RJVI is faster for the randomized and the deterministic cases and appears to be even the best one (transforming the computing times for the used computers) in the randomized $n = 1000$ case in the tests of Dell'Amico and Toth (2000).

The next sections focus on the problems associated with the transformation applications.

3 The *k*-cardinality LAP

The k-LAP with k a given positive integer ($k \leq n$), is the problem to assign only k persons to k jobs, minimizing the total cost, i. e., the k-LAP is a LAP with the extra constraint $\sum_{(i,j)\in A} x_{ij} = k$. Assume $1 \leq k \leq n_1 \leq n_2$ and without loss of generality $c_{ij} > 0$. For ease of presentation, we rearrange the rows of the cost matrix such that the last k rows have the smallest row minima, ρ_i. Denote I as the $n_1 - k$ persons with the largest and *FREE* as the k persons with the smallest row minima.

The (*refined*) transformation of Volgenant (2004) enables to solve the k-LAP as an equivalent RLAP: define $m = n_2$ and add $n_1 - k$ dummy jobs to N_2, with costs (see Fig. 2)

$$\begin{cases} \infty, & \text{for } i \in \{i|1,\ldots,n_1-k\} \text{ and } j = m+1,\ldots,m+n_1-k \text{ and } i \neq j-m, \\ 0, & \text{otherwise.} \end{cases}$$

Now we can initially assign all persons in I to the dummy jobs with zero costs. As the original row minima equal the second lowest row values in the transformed matrix, we can set the node potentials v for all dummy objects j to $v(j) = -\rho_i$, the reverse of the original row minima.

The transformed matrix has a structure that we exploit for an *advanced* transformation, which aims to enable more initial assignments. We first examine the subproblem of the k persons in *FREE*. Clearly, assignments have to be cheaper than assigning one of the first $n_1 - k$ persons. This can be achieved by taking into account the smallest row minimum of these persons, denoted by ρ^+. Assignments at a higher net value than ρ^+ are avoided by adding k dummy jobs with costs $c_{ij} = \rho^+$, for all k persons and $j = m, m+1,\ldots,n_2$. An initial assignment for k-LAP can now be found by performing the procedure AUGROWRED_kBEST (see Appendix B). We can do this more efficiently by adding a single dummy job with a cost $c_{i,n_2+1} = \rho^+$ for each i, and leaving free any person that would be assigned to this dummy job. Consequently, the value v_{m+1} of the dummy must remain 0 all along. After this preprocessing, we delete the dummy jobs and solve the remaining instance by the above refined transformation approach. Since $H()$ and $b()$ are already known for all k persons in FREE, we can continue to use them. However, to assure correctness, $b(i)$ must be set to ρ^+ if $b(i) > \rho^+$.

Dell'Amico and Martello (1997) have developed special k-LAP algorithms based on min-cost flow or successive shortest paths techniques. Their preprocessing determines so-called *fixed* persons and *fixed* jobs that must be assigned in an optimal solution of k-LAP. Experiments showed that this preprocessing failed to give improvements in the transformation approach.

To fairly compare to known results, the largest considered cost range is $[1, 10^5]$ and $n_1 = n_2 = n$. Ten random instances were generated and solved with $n \in \{500, 1000, 1500\}$ and the values $k \in \{0.20, 0.40, 0.60, 0.80, 0.90, 0.99\}n$ (truncated).

We first tested algorithm kRJV, the refined transformation of Volgenant (2004) solved with RJV. We introduce kRJVI (Fig. 3) as the version of RJV that combines the advanced

```
Function kRJVI;
Begin
  ADVANCED_TRANSFORMATION;
  AUGROWRED_6BEST_s;
  If f ≤ 0.1k then AUGMENTATION_s Else AUGMENTATION_P_s;
  Compute optimal value Optimum;
  LAPJVI := Optimum
End.
```

Fig. 3 The algorithm kRJVI

transformation with preprocessing, β-best implementation, partitioning, and exploits the sparsity of the cost matrix. Two implementation details for this code are (1) due to the preprocessing and (2) due to the transformation;

(1) it appeared to be best to do the $k-$best augmenting procedure only once instead of twice;
(2) with f the number of free persons, ratio f/k determines the augmentation procedure to use. It appeared to be best to solve the remaining problem by the existing augmentation procedure, if less than $0.1k$ persons (instead of $0.8n$ in kRJV) are still free after the augmenting row reduction phase.

We exploit the *sparsity* of the cost matrix (Fig. 2) of kRJVI to improve the performance, especially if k/n is much smaller than 1. It is stored whether it is sufficient for a certain person to examine all or only one dummy job. Since assigning a person to a dummy costs 0, it is sufficient to examine the corresponding potentials v. So, fewer elements have to be scanned, which often compensates the related extra work.

The algorithm kRJVI is often 2–4 times faster than kRJV for the uniform random instances (Table 3). The instances with $k = 0.8n$, and $n_1 = 1500$, range $[1, 10^3]$ appear to be harder, maybe as more row minima are equal. Compared to RLAP, the problem is generally harder to solve on larger cost ranges; possibly the β-best implementation is then less effective.

We make a rough comparison with the special algorithm of Dell'Amico and Martello (1997). They concluded for the uniform random class that kLAP instances are much harder to solve than LAP instances, as they observed that their special algorithm is about a factor 5 slower than LAPJV for $k \geq 0.9n$ and cost range $[1, 10^5]$. kRJVI gives a (much) smaller factor: using our fastest algorithms for square LAP instances (see Table 1), LAPJV on range $[1, 10^3]$ and RJVI on cost ranges $[1, 10^2]$ and $[1, 10^5]$, it follows that the kLAP instances are almost always easier to solve on the range $[1, 10^2]$ and indeed harder to solve on the other two ranges. However, the factors are much smaller, from 1.15 to 1.65 for $k = 0.8n$ and $0.9n$; and about 1 for $k = 0.99n$.

The results of kRJV and kRJVI on the Machol-Wien classes (Table 4) vary strongly. We think that the randomized class is hard to solve, because less than 2.5% of the k to be assigned persons are assigned in the augmenting row reduction phase in contrast to at least 96% in RLAP. The best results of the randomized class ($k = 0.2n$ and $0.99n$) are remarkably close.

We make a more refined time comparison of kRJVI to the special algorithm of Dell'Amico and Martello (1997). The results of the Machol-Wien instances (which are exactly the same instances) are obtained about 9 times faster compared to the results of Volgenant (2004), confirmed by averaging the results of the refined transformation for the

Table 3 The results for the random k-LAP instances; kRJV and kRJVI times in ms

k	n_1 & n_2	Range $[1, 10^2]$			Range $[1, 10^3]$			Range $[1, 10^5]$		
		kRJV	kRJVI	ϕ in %	kRJV	kRJVI	ϕ in %	kRJV	kRJVI	ϕ in %
$0.2n_1$	500	22	2	90.9	22	6	72.7	21	7	66.7
	1000	75	21	72.0	85	28	67.1	87	23	73.6
	1500	169	46	72.8	195	55	71.8	205	60	70.7
$0.4n_1$	500	20	7	65.0	25	13	48.0	26	12	53.8
	1000	72	29	59.7	88	38	56.8	133	52	60.9
	1500	156	69	55.8	402	249	38.1	404	127	68.6
$0.6n_1$	500	25	10	60.0	31	19	38.7	35	23	34.3
	1000	73	40	45.2	259	151	41.7	250	127	49.2
	1500	162	105	35.2	316	153	51.6	828	332	59.9
$0.8n_1$	500	43	19	55.8	34	26	23.5	47	36	23.4
	1000	102	54	47.1	238	164	31.1	333	263	21.0
	1500	172	123	28.5	727	508	30.1	1047	804	23.2
$0.9n_1$	500	32	22	31.3	33	25	24.2	44	38	13.6
	1000	185	62	66.5	183	169	7.7	301	296	1.7
	1500	271	139	48.7	522	376	28.0	930	864	7.1
$0.99n_1$	500	25	26	4.0	36	33	8.3	41	34	17.1
	1000	128	102	20.3	159	145	8.8	260	236	9.2
	1500	257	171	33.5	389	353	9.3	699	656	6.2
Average				*49.1*			*36.5*			*36.7*

Table 4 The entrances are ratios of times of kRJVI and the algorithm of Dell'Amico and Martello (1997); a ratio >1 indicates that kRJVI is faster

n_1 & n_2	k	Uniform class			Machol Wien	
		$[1, 10^2]$	$[1, 10^3]$	$[1, 10^5]$	Randomized	Standard
500	$0.2n_1$	4.00	1.17	3.14	0.48	0.19
	$0.4n_1$	1.14	2.31	3.00	0.55	0.35
	$0.6n_1$	0.80	2.16	2.48	0.65	0.77
	$0.8n_1$	2.11	2.68	3.19	1.88	1.43
	$0.9n_1$	1.95	3.60	3.86	2.17	1.99
	$0.99n_1$	2.00	3.90	6.14	1.63	2.77

uniform class with $n = 500$. In Volgenant (2004) the results have been obtained on an AMD K6 333 MHz processor, indicating a ratio of about 3 to the personal computer used in our research; Volgenant (2004) assumed that these results are obtained about a factor 18 faster than the special algorithm. We think that the additional speed up is due to the programming language (Delphi 6 versus Turbo Pascal) and to the allocations of the data structures (static versus more dynamic pointers).

Thus, to fairly although roughly compare the solution times of the algorithms to each other we divided all computing times of the special algorithm by a factor of 162 ($= 9 \times 18$) and rounded them to integers.

Table 5 The results for the Machol-Wien k-LAP instances; kRJV and kRJVI times in ms

k	n_1 & n_2	Randomized			Standard		
		kRJV	kRJVI	ϕ in %	kRJV	kRJVI	ϕ in %
$0.2n_1$	500	179	58	67.6	550	246	55.3
	1000	1458	404	72.3	4219	1331	68.5
	1500	4722	1098	76.7	14039	3715	73.5
$0.4n_1$	500	382	100	73.8	970	490	49.5
	1000	2792	846	69.7	7590	2963	61.0
	1500	9469	2752	70.9	25131	8531	66.1
$0.6n_1$	500	414	129	68.8	1275	682	46.5
	1000	3460	1140	67.1	9751	4564	53.2
	1500	11677	3553	69.6	32688	13555	58.5
$0.8n_1$	500	265	108	59.2	1361	837	38.5
	1000	2789	1050	62.4	10694	5717	46.5
	1500	9533	3255	65.9	35578	17416	51.0
$0.9n_1$	500	205	93	54.6	1277	852	33.3
	1000	1966	740	62.4	10440	6020	42.3
	1500	6493	2631	59.5	35087	18540	47.2
$0.99n_1$	500	104	96	7.7	1189	850	28.5
	1000	745	571	23.4	10059	6076	39.6
	1500	1845	1237	33.0	33445	18782	43.8
Average				*59.1*			*50.2*

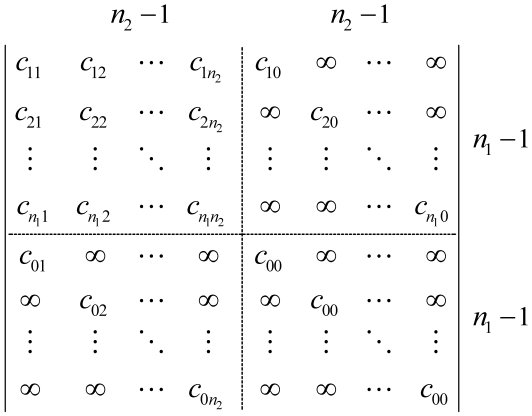
Table 5 shows this comparison between the special algorithms and kRJVI for $n = 500$. On these problem instances, kRJVI is much faster for almost all uniform random instances. The rather large ratio of $k = 0.2n$ and the range $[1, 10^2]$ may be due to the small computation times, making them less precise. For the Machol-Wien instances the special algorithm is faster if $k \leq 0.6n$ and kRJVI is faster if $k > 0.6n$. We finally note that kRJVI is easier to implement than the special algorithm.

4 The LAP with outsourcing

To allow for the alternative of sourcing or contracting out internal jobs to external machines (or persons) and processing external jobs on internal machines, Wiel Vander and Sahinidis (1997) considered the assignment problem with external interactions (APEX). They formulated it as a LAP with a single unrestricted variable and developed a special primal-dual algorithm. The mathematical model maintains the problem's special structure and its size:

$$
\begin{aligned}
\text{(APEX)} \quad \text{Minimize} \quad & \textstyle\sum_{(i,j)\in A} c_{ij}x_{ij} \\
\text{subject to} \quad & \textstyle\sum_{j\in N_2(i)} x_{ij} = 1, \quad i \in N_1 \\
& \textstyle\sum_{i\in N_1(j)} x_{ij} = 1, \quad j \in N_2 \\
& x_{ij} \geq 0, \quad (i,j) \in \mathcal{A}\backslash(i^\circ, j^\circ) \\
& x_{i^\circ j^\circ} \in Z
\end{aligned}
$$

Fig. 4 The cost matrix of the LAPout transformation



$$\begin{array}{c|ccc|ccc|c}
c_{11} & c_{12} & \cdots & c_{1n_2} & c_{10} & \infty & \cdots & \infty & \\
c_{21} & c_{22} & \cdots & c_{2n_2} & \infty & c_{20} & \cdots & \infty & \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & n_1-1 \\
c_{n_1 1} & c_{n_1 2} & \cdots & c_{n_1 n_2} & \infty & \infty & \cdots & c_{n_1 0} & \\
\hline
c_{01} & \infty & \cdots & \infty & c_{00} & \infty & \cdots & \infty & \\
\infty & c_{02} & \cdots & \infty & \infty & c_{00} & \cdots & \infty & \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & n_1-1 \\
\infty & \infty & \cdots & c_{0n_2} & \infty & \infty & \cdots & c_{00} &
\end{array}$$

where i° represents all external persons to which jobs can be assigned to and j° represents all external jobs to which persons can be assigned to. Each internal person can either be assigned to an available internal job or outsourced to an external job. Similarly, each internal job can either be assigned to an available internal person or outsourced to an external person. We assume that $|N_1(j^\circ)| = |N_2(i^\circ)| = n_1 - 1$. Hence, in any feasible solution of APEX, arc (i°, j°) has a flow from j° to i° that is one less than the number of persons (or jobs) that are assigned externally. If no person is assigned externally, there is a unit flow from i° to j°. In general the flow on arc (i°, j°) is implicitly bounded by the other constraints: $-(n-2) \le x_{i^\circ j^\circ} \le 1$.

We can solve the APEX by standard LAP algorithms, while keeping the special structure of APEX. We replace person i° by $n_1 - 1$ persons and job j° by $n_2 - 1$ jobs, and duplicate all the associated arcs. We refer to the resulting model as LAPout, the LAP with outsourcing. In the related cost matrix (Fig. 4) the external persons and jobs have 0 as index. An optimal solution of LAPout is related to an optimal solution of APEX: set $x_{i^\circ j^\circ} = x_{i^\circ j^\circ} - (n_1 - 2)$. The optimal values z_A of APEX and z_L of LAPout are related by $z_A = z_L - (n_1 - 2)c_{00}$.

A drawback of LAPout is that it increases the problem size. To overcome this Wiel Vander and Sahinidis (1997) reformulated APEX as the *full assignment problem* (FAP) that can be solved by any standard LAP algorithm. While FAP does not increase the problem size, it ruins the (special) structure of the cost matrix, especially the sparsity:

$$\begin{array}{lll}
\text{(FAP)} & \text{Minimize} & \sum_{i\in N_1\setminus i^\circ} \sum_{j\in N_2\setminus j^\circ} \min\{c_{ij}, c_{i0} - c_{00} + c_{0j}\} x_{ij} \\
& \text{subject to} & \sum_{j\in N_2(i)\setminus j^\circ} x_{ij} = 1, \quad i \in N_1\setminus j^\circ \\
& & \sum_{i\in N_1(j)\setminus i^\circ} x_{ij} = 1, \quad j \in N_2\setminus j^\circ \\
& & x_{ij} \ge 0, \quad i \in N_1\setminus i^\circ, j \in N_2\setminus j^\circ
\end{array}$$

If $x_{ij} = 1$ in an optimal solution of FAP, then it is also in an optimal solution of APEX if $c_{ij} = \min\{c_{ij}, c_{i0} - c_{00} + c_{0j}\}$; otherwise this solution contains $x_{ij^\circ} = x_{i^\circ j^\circ} = x_{i^\circ j} = 1$. The optimal values z_A and z_F of APEX and FAP are related by $z_A = z_F + c_{00}$.

Alternatively, we can reduce the size of the cost matrix of LAPout by transforming it into the matrix of a problem denoted as LAPout', while maintaining the special structure as follows: Subtract first c_{00} from the last $n_1 - 1$ rows, then $c_{0j} - c_{00}$ from the first $n_2 - 1$ columns and finally c_{i0} from the first $n_1 - 1$ rows. As the costs in the last $n_1 - 1$ rows are

$$\left|\begin{array}{cccc|cccc} & n_2-1 & & & & n_2-1 & & \\ c_{11}-c_{01}-c_{10}+c_{00} & c_{21}-c_{02}-c_{10}+c_{00} & \cdots & c_{1n_2}-c_{0n_2}-c_{10}+c_{00} & 0 & \infty & \cdots & \infty \\ c_{21}-c_{01}-c_{20}+c_{00} & c_{22}-c_{02}-c_{20}+c_{00} & \cdots & c_{2n_2}-c_{0n_2}-c_{20}+c_{00} & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n_1 1}-c_{01}-c_{n_1 0}+c_{00} & c_{n_1 2}-c_{02}-c_{n_1 0}+c_{00} & \cdots & c_{n_1 n_2}-c_{0n_2}-c_{n_1 0}+c_{00} & \infty & \infty & \cdots & 0 \end{array}\right| \begin{array}{l} \\ \\ \\ n_1-1 \end{array}$$

Fig. 5 The rectangular cost matrix of LAPout after transformation (LAPout')

0 or ∞, it makes no difference to assign these external persons to an internal job or to an external one; thus it is sufficient to solve an RLAP on the LAPout' matrix (Fig. 5).

The following theorem shows the similarity between APEX and LAPout':

Theorem 1 *An optimal solution of the RLAP solved on the LAPout' matrix is also optimal for the APEX problem and vice versa.*

Proof First note the similarity between LAPout' and FAP. If $x_{ij} = 1$ in an optimal solution of LAPout' then $c_{ij} \leq 0$; otherwise $x_{ij^\circ} = 1$, as that improves the criterion value. Clearly, the opposite, if $x_{ij^\circ} = 1$ then $c_{ij} > 0$ also holds.

Thus, if $x_{ij} = 1$ in an optimal solution of LAPout', then $c_{ij} \leq c_{i0} - c_{00} + c_{0j}$; so it is also in an optimal solution of APEX, like in FAP. If on the other hand $x_{ij^\circ} = 1$ in an optimal solution of LAPout', then for any unassigned internal job j, $c_{ij} > c_{i0} - c_{00} + c_{0j}$; otherwise outsourcing person i would not be optimal. Thus, $x_{ij^\circ} = x_{i^\circ j^\circ} = x_{i^\circ j} = 1$ in an optimal solution of APEX.

The optimal value z_F of FAP and $z_{L'}$ of LAPout' are related: z_F follows by adding up $z_{L'}$ and $c_{i0} - c_{00} + c_{0j}$ $\forall (i, j) \in X$, with X an optimal solution of FAP; so

$$\begin{aligned} z_A &= z_{L'} + c_{00} - (n_1 - 1)c_{00} + \sum_{i \in N_1 \setminus i^\circ} c_{i0} + \sum_{j \in N_2 \setminus j^\circ} c_{0j} \\ &= z_{L'} + \sum_{i \in N_1 \setminus i^\circ} c_{i0} + \sum_{j \in N_2 \setminus j^\circ} c_{0j} - (n_1 - 2)c_{00}. \end{aligned}$$

□

The structure of the LAPout' matrix (Fig. 5) allows less preprocessing than in the case of k-LAP. It even appeared that preprocessing of LAPout' gave no better computational results.

In Wiel Vander and Sahinidis (1997) the special algorithm has been compared among others against FAP solved by LAPJV, mainly on uniform random test instances with external arcs $c_{00} = 0$. We focus on the influence of different values of these arcs ($c_{00} \in [0, \ldots, 2]R$) on the computing times, comparing the transformation LAPout' to FAP for dense instances. Ten uniform random instances were generated and solved with different values of c_{00}, each range and $n = 500$, 750 or 1000, and cost ranges $[1, R]$ with $R \in [10^2, 10^3, 10^5]$; the external arcs have costs 0 and R. The times (ms) are averages of the ten instances.

Table 6 gives the computational results. FAP is solved by LAPJV as well as by RJVI, denoted as FAP and FAPI respectively. RJVoutI is LAPout' solved by RJVI. To reduce the number of elements to be scanned and the number of elements to be stored, we exploit the structure of LAPout'. Because there is only one relevant external job for each person, a person is outsourced if the cost of the selected (cheapest) assignment is non-negative.

Table 6 Computing times (ms) for LAP with outsourcing, $c_{00} = 0$ or R

n	Range	$c_{00} = 0$			$c_{00} = R$		
		FAP	FAPI	RJVoutI	FAP	FAPI	RJVoutI
500	$[1, 10^2]$	30	27	333	95	219	65
	$[1, 10^3]$	25	33	531	152	329	100
	$[1, 10^5]$	36	28	533	207	356	80
750	$[1, 10^2]$	92	76	905	341	673	217
	$[1, 10^3]$	89	92	1807	570	1018	360
	$[1, 10^5]$	105	95	1960	816	1143	311
1000	$[1, 10^2]$	148	131	1637	797	1458	499
	$[1, 10^3]$	173	178	4215	1420	2178	869
	$[1, 10^5]$	215	208	4750	1875	2608	770

As in the kRJVI algorithm, the sparsity is exploited in both augmentation procedures. The LAPout' transformation is clearly less efficient than the FAP algorithms for instances with $c_{00} = 0$, the opposite holds for instances with $c_{00} = R$, while FAPI performs poorly on these instances.

In FAP we set $c_{ij} = \min\{c_{ij}, c_{i0} + c_{0j} - c_{00}\}$, so for large c_{00}-values the second term is often dominating. FAPI starts at once with augmenting row reduction; as a result, for each row the first and the second row minimum often occur at the same jobs; $j_1 = \arg\min_{j \in N_2}\{c_{0j}\}$ and $j_2 = \arg\min_{j \in N_2 \setminus \{j_1\}}\{c_{0j}\}$. Therefore, reduction transfer often fails, and many persons must be assigned in the more time-consuming augmentation phase. FAP, however, begins with column reduction, in which the node potentials v are given values. As a result, the first and the second row minima less often occur at the same jobs in the augmenting row reduction part.

After the LAPout' transformation, $c_{ij} := c_{ij} - c_{i0} - c_{0j} + c_{00}$. It is likely that row minima occur at similar jobs in the case that c_{00} is small. Though, in this case reduction transfer does take place, but only by small values. When c_{00} is larger, the external job can be more often the row minimum, making it easier to assign the remaining persons.

The results of FAP, FAPI and RJVoutI (Fig. 6) with $c_{00} = 0, 20, \ldots, 200$ and cost range $[1, 10^2]$ show that FAPI is the fastest code if $c_{00} \leq 0.65R$, and RJVoutI otherwise. Similar results can be shown with other values of n and other cost ranges. We can conclude that the two algorithms complement each other.

5 The (k-)LAP for replacement

Suppose the manager of a plant has the funds to replace k among the n_2 machines, by newer ones. He wants to optimally assign tasks to machines under maximal production or equivalent, minimal production cost. Caseau and Laburthe (2000) denoted this problem as the replacing k machines linear assignment problem (k-LAPrep), given that new machines are more efficient than older ones. They assume to replace exactly k machines, even if this gives worse results. We formulate the more general case omitting the latter restriction and where old machines may be *more* or *less* efficient than new ones. We develop a transformation that enables to solve it as a RLAP.

Let the (production) cost of processing task i be c_{ij} on machine j and d_{ij} on a replaced machine j. Assume without loss of generality $1 \leq k \leq n_1 \leq n_2$ and positive costs. Define

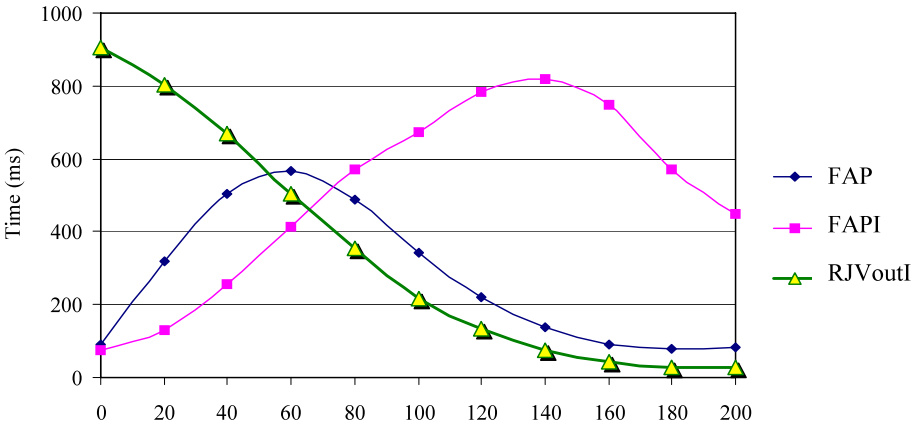


Fig. 6 Results for LAP with outsourcing, $n = 750$, and $0 \le c_{00} \le 200$

$m_2 = n_2$ and add m_2 machines to N_2 at cost d_{ij}; thus we have $n_2 = 2m_2$ and $c_{i,m_2+j} = d_{ij}$, $i \in N_1$, $j \in N_2$ and we can formulate k-LAPrep as an RLAP with one extra constraint:

$$\sum_{i=1}^{n_1} \sum_{j=m_2+1}^{n_2} x_{ij} \le k$$

We transform k-LAPrep into an RLAP denoted as $k-$LAPrep', defining $m_1 = n_1$ and we add $m_2 - k$ dummy nodes to N_1, at cost:

$$c_{ij} = \begin{cases} 0, & \text{for } i = m_1 + 1, \ldots, n_1 \text{ and } j = m_2 + 1, \ldots, n_2, \\ \infty, & \text{otherwise,} \end{cases}$$

with $n_1 = m_1 + m_2 - k$. Theorem 2 is proven similarly as the simple k-LAP transformation in Volgenant (2004).

Theorem 2 *The problems k-LAPrep' and k-LAPrep are equivalent.*

Proof A solution of k-LAPrep can be extended to a solution of k-LAPrep' with the same criterion value, by adding $m_2 - k$ (zero cost) slack assignments. There are many equivalent solutions for k-LAPrep', thus many solutions of k-LAPrep' correspond to one solution of k-LAPrep, to be found by deleting the slack assignments of the k-LAP' solution.

Let S^* be an optimal solution of k-LAPrep with criterion value $z(S^*) = z^*$; as shown, it can be extended to a feasible solution S_e of k-LAPrep', with the same criterion value $z'(S_e) = z^*$. Clearly S_e is also an optimal solution of k-LAPrep'.

Suppose S' is an optimal solution of k-LAPrep' with criterion value $z'(S') = z'$. Then S' contains $m_2 - k$ slack assignments of 0 cost, because all non-slack c-values are positive. Therefore, S' contains at most k, say h ($h \le k$), non-slack assignments to new machines (i.e., $i = 1, \ldots, m_1$ and $j = m_2 + 1, \ldots, n_2$) and at least $m_1 - h$ non-slack assignments to old machines (i.e., $i = 1, \ldots, m_1$ and $j = 1, \ldots, n_1$). Deleting the slack assignments of S' forms a feasible solution, say S'', for k-LAPrep and $z(S'') = z(S')$. Now $z'(S') > z^* = z'(S_e)$ contradicts the optimality of S', i.e., $z'(S') = z'(S_e)$. Thus S'' is an optimal solution of k-LAPrep; i.e., $z(S'') = z(S^*)$ and the problems k-LAPrep' and k-LAPrep are equivalent. □

$$
\begin{array}{cccc|cccccccc|c}
 & & m_2 & & & & & & & & m_2 & & \\
c_{11} & c_{12} & \cdots & c_{1m_2} & d_{11} & d_{12} & \cdots & d_{1,n_2-k} & \cdots & \cdots & d_{1m_2} & \\
c_{21} & c_{22} & \cdots & c_{2m_2} & d_{21} & d_{22} & \cdots & d_{2,n_2-k} & \cdots & \cdots & d_{2m_2} & \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & m_1 \\
c_{m_1 1} & c_{m_1 2} & \cdots & c_{m_1 m_2} & d_{m_1 1} & d_{m_1 2} & \cdots & d_{m_1,n_2-k} & \cdots & \cdots & d_{m_1 m_2} & \\
\hline
\infty & \infty & \cdots & \infty & 0 & \infty & \cdots & \infty & 0 & \cdots & 0 & \\
\infty & \infty & \cdots & \infty & \infty & 0 & \cdots & \infty & 0 & \cdots & 0 & m_2-k \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \\
\infty & \infty & \cdots & \infty & \infty & \infty & \cdots & 0 & 0 & \cdots & 0 &
\end{array}
$$

Fig. 7 The cost matrix of k-LAPrep after transformation (k-LAPrep')

It is efficient, see Volgenant (2004), to avoid degeneracy in the set of feasible solutions of a problem, by replacing the zero elements of k-LAPrep' by ∞ in the rows

$$i = m_1 + 1, \ldots, n_1 \quad \text{and} \quad j = m_2 + 1, \ldots, n_2 - k \quad \text{and} \quad j \neq i + m_2 - m_1$$

resulting in the matrix of Fig. 7.

This cost matrix does not necessarily include or exclude any assignments. Yet, it is easy to apply preprocessing, as we can assign all dummy tasks i $(i = m_1 + 1, \ldots, n_1)$ to the first $m_2 - k$ new machines j, at 0 cost, i.e., assign dummy task i to machine $j = m_2 + i - m_1$ and solve the remaining problem by an adapted RJV algorithm (denoted as RLAPrepI).

Exploiting efficiently the sparsity of the cost matrix instead of actually transforming that matrix, it is sufficient to check whether the chosen task is a real or a dummy one. For each dummy, say d, one only has to scan the node potentials of the jobs

$$j \in \{j \mid j = d + m_2 - m_1 \text{ or } j = n - k + 1, \ldots, n\}.$$

Table 7 gives computational results on the realistic case that new machines produce cheaper than old ones, although the production costs on new and old machines can be arbitrary. We generated uniform random production costs with c_{ij} for processing task i on old machine j $(i = 1, \ldots, n_1,\ j = 1, \ldots, m_2)$ and with costs in range $[1, c_{ij}]$ for processing task i on a new (replaced) machine j $(i \in N_1,\ j \in N_2)$. We generated 10 instances for each value of $n_1 = n_2 = n$ and cost ranges $[1, R]$, $R \in \{10^2, 10^3, 10^6\}$, with the number of new as well as old machines $n \in \{500, 750, 1000\}$ and $k \in \{0.01, 0.1, 0.25, 0.50, 0.75, 0.90, 0.99\}n$.

Caseau and Laburthe (2000) gave computational results only for instances up to size 40, as the logic constraint approach fails for larger sizes. Since we know of no other (special) code for k-LAPrep we demonstrate the effect of the improvements by comparing 2 codes. The code denoted by kRJVrep transforms k-LAPrep to kLAPrep' and solves the remaining problem with RJV. In the transformed matrix $n = 2m_2$, the numbers of machines and of tasks have chosen to be equal in the considered problem instances, i.e., $m_2 = m_1$.

Although the new machines produce tasks cheaper than the old ones ($c_{ij} > d_{ij}$), it is not always best to replace all the k machines. It appears that up to $k = 0.75n$ almost all

Table 7 Computing times (in ms) for kLrep(I) with more efficient new machines (L short for LAP)

k	n	Range $[1, 10^2]$			Range $[1, 10^3]$			Range $[1, 10^6]$		
		kLrep	kLrepI	ϕ in %	kLrep	kLrepI	ϕ in %	kLrep	kLrepI	ϕ in %
$0.01n$	500	180	84	53.3	352	173	50.9	682	394	42.2
	750	532	197	63.0	863	399	53.8	2006	1228	38.8
	1000	855	217	74.6	1655	678	59.0	4489	2503	44.2
$0.1n$	500	375	61	83.7	599	227	62.1	1460	876	40.0
	750	1131	95	91.6	1949	588	69.8	4931	2927	40.6
	1000	2513	164	93.5	4426	999	77.4	11806	7131	39.6
$0.25n$	500	446	61	86.3	726	289	60.2	1570	1047	33.3
	750	1517	128	91.6	2435	713	70.7	5329	3591	32.6
	1000	3662	223	93.9	5617	1221	78.3	12366	8408	32.0
$0.5n$	500	211	30	85.8	509	214	58.0	983	754	23.3
	750	772	69	91.1	1570	395	74.8	3290	2475	24.8
	1000	2003	104	94.8	3197	379	88.1	7628	5788	24.1
$0.75n$	500	23	14	39.1	97	47	51.5	226	159	29.6
	750	125	89	28.8	215	102	52.6	781	568	27.3
	1000	360	145	59.7	331	174	47.4	1794	1290	28.1
$0.9n$	500	17	12	29.4	22	18	18.2	20	9	55.0
	750	41	32	22.0	46	37	19.6	50	30	40.0
	1000	106	110	−3.8	79	61	22.8	90	50	44.4
$0.99n$	500	11	13	−18.2	18	16	11.1	18	11	38.9
	750	35	30	14.3	37	33	10.8	40	23	42.5
	1000	90	85	5.6	65	52	20.0	74	43	41.9
Average				*56.2*			*50.3*			*36.4*

the k machines are actually replaced and that for higher values of k the replacement percentage drops to a minimum of 72%. The average improvement factor ϕ of kRJVrepI is 47.6 % and the results are less sensitive to hard instances, i.e., ϕ is larger than average for most of the hardest instances. Averaging over the times, kRJVrepI is more than 3 times faster than kRJVrep. Instances with $k = 0.25n$ are on average the hardest to solve.

6 Summary and conclusions

We considered the rectangular LAP, i. e., the number of persons differs from the number of jobs and improved an existing algorithm to solve it more efficiently, resulting in the algorithm RJVI. As RJVI can detect hard problem instances during the execution of the algorithm, it can solve such instances faster by partitioning the job set in assigned and in free jobs. The algorithm RJVI also uses a β-best implementation, which appears to increase robustness over various cost ranges. It is able to efficiently solve both rectangular and square LAP instances.

We have described and implemented preprocessing by exploiting the structure of the cost matrices to further improve the performance of RJVI on the transformation applications

kLAP, LAPout and kLAPrep. The special structures of the cost matrices allow additional improvements, which enable to solve these applications efficiently, using general and easy to implement codes. The computational results showed that one can efficiently solve the three considered variants of the LAP by applying suited transformations and that our codes are often even much faster than existing special algorithms.

We conclude that solving the considered LAP variants by means of transformations to a RLAP appears to be flexible and efficient.

Acknowledgements We thank the Editor-in-Chief Professor Boros for useful remarks and we are grateful to an anonymous reviewer who suggested changes that have improved the paper.

Appendix A: The augmentation part with $N2(i)$ partitioned into $P(i)$ and $Q(i)$

```
Procedure AUGMENTATION_P;
Begin
  for all free i* do
  Begin
    INITIALIZE;
    repeat
      if SCAN={} then
      Begin
        SELECT_OBJECTS;
        if d[j] <= μ; then goto augment
      End;
      select any j* ∈ SCAN; i := y[j*];
      SCAN:=SCAN-{j*}; READY:=READY+{j*};
      u1 := c[i, j*] − v[j*] − μ;
      EXAMINE_Q;
      if h = μ then goto augment;
      EXAMINE_P;
    until false;(*repeat ends with go to augment*)
  End;
augment:
      for all k ∈ READY do v[k] := v[k] + d[k] − μ; (*price updating*)
      repeat (*augmentation*)
        i :=pred[j]; y[j] := i; k := j; j := x[i]; x[i] := k;
      until i = i*;
  End
End;

Procedure INITIALIZE;
Begin
  buildheap(Q[i*]); j := Q[i*, 1]; pred[j] := i*; d[j] := c[i*, j];
  READY:={}; SCAN:={}; TODOa:= {all assigned columns}; TODOu:={j};
  UNLAB:={all unassigned columns}-{j};
  for all j ∈TODOa do Begin d[j] := c[i*, j] − v[j]; pred[j] := i* End
End;

Comment all possible job sets during the augmentation phase are mutually disjoint and
        READY ∪ SCAN ∪ TODOa ∪ TODOb ∪ UNLAB = N;
```

```
Procedure SELECT_OBJECTS;
Begin
  μ := min{d[j]|j ∈TODOa}; SCAN:={j|d[j] = μ}; TODOa:=TODOa-SCAN;
  j := min{d[k]|k ∈TODOu};
End;

Procedure EXAMINE_Q;
Begin
  if Q[i] = {} then buildheap(Q[i])
  else if Q[i, 1] is assigned then
    repeat delete(Q[i, 1]) until Q[i, 1] is free;
  j := Q[i, 1]; h := c[i, j] − u1;
  if h = μ then pred[j] :=i
  else if j ∈UNLAB then
    Begin d[j] := h; pred[j] := i; TODOu:=TODOu+{j}; UNLAB:=UNLAB-{j} End
  else if h < d[j] then Begin h := d[j]; pred[j] := i End
End;

Procedure EXAMINE_P;
Begin
  for all j ∈TODOa do if c[i, j] − v[j] − u1 < d[j] then
    Begin
      d[j] := c[i, j] − v[j] − u1; pred[j] := i;
      if d[j] = μ then Begin SCAN:=SCAN+{j}; TODO:=TODO-{j} End
    End
End.
```

Appendix B: The augmenting row reduction part with the *k*-best implementation

```
Procedure AUGROWRED_kBEST;
Begin
  FREE:={all free persons}; b[i] := {}; H[i] := {};
  for all i ∈FREE do
  repeat
    if b[i] ≠ {} then
      Begin
        j := 0;
        for all k ∈ H[i] do
          if c[i, k] − v[k] ≤ b[i] then j := j + 1;
        if j < 2 then recompute H[i] and b[i];
      End
    else compute H[i] and b[i];
    u1 := min{c[i, j] − v[j]|j ∈ H[i]};
    select j1 with c[i, j1] − v[j1] = u1;
    u2 := min{c[i, j] − v[j]|j ∈ H[i], j <> j1};
    select j2 with c[i, j2] − v[j2] = u2;
    if u1 < u2 then v[j1] := v[j1] − (u2 − u1)
      else if j1 is assigned then j1 := j2
    k := y[j1]; if k > 0 then x[k] := 0;
    x[i] := j1; y[j1] := i; i := k;
  until u1 = u2 (*no reduction transfer*) or k = 0 (*augmentation*)
End.
```

References

Bertsekas, D. P., & Castañon, D. A. (1992). A forward reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications*, *1*, 277–297.

Bertsekas, D. P., Castañon, D. A., & Tsaknakis, H. (1993). Reverse auction and the solution of asymmetric assignment problems. *SIAM Journal on Optimization*, *3*, 268–299.

Burkard, R., Dell'Amico, M., & Martello, S. (2009). *Assignment Problems*. Philadelphia: Society for Industrial and Applied Mathematics.

Caseau, Y., & Laburthe, F. (2000). Solving weighted matching problems with constraints. *Constraints, an International Journal*, *5*, 141–160.

Dell'Amico, M., & Martello, S. (1997). The k-cardinality assignment problem. *Discrete Applied Mathematics*, *76*, 103–121.

Dell'Amico, M., & Toth, P. (2000). Algorithms and codes for dense assignment problems: the state of the art. *Discrete Applied Mathematics*, *100*, 17–48.

Goldberg, A. V., & Kennnedy, J. R. (1995). An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, *71*, 153–177.

Hsieh, A. J., Fan, K. C., & Fan, T. I. (1995). Bipartite weighted matching for on-line handwritten Chinese character recognition. *Pattern Recognition*, *28*, 143–151.

Jonker, R., & Volgenant, A. (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, *38*, 325–340.

Knuth, D. E. (1981). *The Art of Computer Programming*, Vol. 2: *Seminumerical Algorithms*. Reading: Addison-Wesley.

Machol, R. E., & Wien, M. (1976). A hard assignment problem. *Operations Research*, *24*, 190–192.

Mosheiov, G., & Yovel, U. (2006). Minimizing weighted earliness-tardiness and due-date cost with unit processing-time jobs. *European Journal of Operational Research*, *172*, 528–544.

Pinedo, M. (2002). *Scheduling Theory, Algorithms, and Systems* (2nd edn.). Upper Saddle River: Prentice Hall.

Wiel Vander, R. J., & Sahinidis, N. V. (1997). The assignment problem with external interactions. *Networks*, *30*, 171–185.

Volgenant, A. (1996). Linear and semi-assignment problems, a core oriented approach. *Computers & Operations Research*, *10*, 917–932.

Volgenant, A. (2004). Solving the k-cardinality assignment problem by transformation. *European Journal of Operational Research*, *157*, 322–331.