Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem

Snežana Mitrović-Minić, Abraham P. Punnen [*]

Department of Mathematics, Simon Fraser University, BC, Canada

ARTICLE INFO

Article history:
Received 7 October 2006
Received in revised form 17 December 2008
Accepted 26 April 2009
Available online 19 May 2009

Keywords:
Generalized assignment
0–1 integer programs
Heuristics
Local search
VLSN Search
Variable neighbourhood search
Resource allocation

ABSTRACT

We introduce a heuristic for the Multi-Resource Generalized Assignment Problem (MRGAP) based on the concepts of Very Large-Scale Neighborhood Search and Variable Neighborhood Search. The heuristic is a simplified version of the Very Large-Scale Variable Neighborhood Search for the Generalized Assignment Problem. Our algorithm can be viewed as a k-exchange heuristic; but unlike traditional k-exchange algorithms, we choose larger values of k resulting in neighborhoods of very large size with high probability. Searching this large neighborhood (approximately) amounts to solving a sequence of smaller MRGAPs either by exact algorithms or by heuristics. Computational results on benchmark test problems are presented. We obtained improved solutions for many instances compared to some of the best known heuristics for the MRGAP within reasonable running time. The central idea of our heuristic can be used to develop efficient heuristics for other hard combinatorial optimization problems as well.

1. Introduction

The *Multi-Resource Generalized Assignment Problem* (MRGAP) deals with assigning tasks (jobs) to agents (machines) subject to multi-resource constraints for each agent. The MRGAP is a generalization of the *Generalized Assignment Problem* (GAP).

Let $N = \{1, 2, \dots, n\}$ be a set of tasks, $M = \{1, 2, \dots, m\}$, $m \le n$, be a set of agents, and $U = \{1, 2, \dots, u\}$ be a set of resources. Each agent i has capacity b_{il} of resource l. Processing task j by agent i takes a_{ijl} units of resource l from the agent's capacity b_{il}, and costs c_{ij} units. Each task has to be assigned to exactly one agent, while each agent can process several tasks subject to its capacity restrictions. Then the MRGAP seeks a minimum cost assignment of tasks to agents satisfying the agent capacity constraints.

The MRGAP can be formulated as a 0–1 *integer programming problem* as follows:

$$
\begin{aligned}
\textit{MRGAP}: \quad \text{Minimize} \quad & z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{Subject to} \quad & \sum_{j=1}^{n} a_{ijl} x_{ij} \le b_{il}, \quad i = 1, 2, \dots, m; \;\; l = 1, 2, \dots, u \\
& \sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, 2, \dots, n \\
& x_{ij} \in \{0, 1\}, \quad i = 1, 2, \dots, m; \;\; j = 1, 2, \dots, n,
\end{aligned}
$$

where the decision variable x_{ij} takes value one when task j is assigned to agent i. Note that an MRGAP solution $x = [x_{ij}]_{i \in M, j \in N}$ is represented as a matrix.

[*] Corresponding address: Department of Mathematics, Simon Fraser University, 13450-102 Avenue, Surrey, BC, Canada, V3T 5X3.
E-mail address: apunnen@sfu.ca (A.P. Punnen).

1572-5286/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2009.04.004

The MRGAP, being a generalization of the GAP, is NP-hard even for $m = 2$. The model has many practical applications in distributed computer systems and in the tracking industry [1–4]. Reviews of applications and solution approaches to the GAP can be found in the survey papers [5,6].

Various types of exact algorithm are available for the GAP [7–9], primarily based on branch and bound or column generation. The heuristics developed for the GAP are mostly of local search type[10,11] and its variations [12–14, 6,15,33]. Other notable heuristic approaches include linear relaxation [16,17], set partitioning [18], and Lagrangian decomposition [19]. However, unlike the GAP, the MRGAP is not very well studied.

For the MRGAP, Gavish and Prikul proposed a branch and bound algorithm and two simple Lagrangian heuristics [20]. Recently, Yagiura et al. [21] developed a very large-scale neighborhood (VLSN) search heuristic. They also generated a set of MRGAP test instances by extending the GAP instances from the well-known benchmarks containing C, D and E instance types with 100 and 200 tasks[22,21].

In this paper, we introduce an efficient local search heuristic for the MRGAP drawing ideas from the VLSN search [23, 24] and variable neighborhood (VN) search [25]. The heuristic is a simplified version of the Very Large-Scale Variable Neighborhood (VLSVN) search we introduced in [26]. The algorithm searches k-exchange neighborhoods with large k. A large value of k increases the possibility of finding a higher-quality solution in the neighborhood, but at the same time increases the computational burden. To handle this increased complexity, we use an approximate search of the k-exchange neighborhood. Our heuristic either outperformed the best known heuristic [21] by achieving better-quality solutions for the benchmark problems or achieved the same quality solutions with only few exceptions.

The paper is organized as follows. In Section 2 we discuss our k-exchange neighborhood and develop methods to search the neighborhood using heuristic techniques. Also, we discuss our heuristic algorithm. Section 3 deals with computational results, while concluding remarks are provided in Section 4.

2. Solution methodology

Let $x = (x_{ij})_{m\times n}$ be a solution to the MRGAP; i.e.

$$x_{ij} = \begin{cases} 1, & \text{if task } j \text{ is processed by agent } i \\ 0, & \text{otherwise.} \end{cases}$$

We call x a *task–agent assignment* or simply an *assignment*. For each $i = 1, 2, \ldots, m$, let $Q_x(i)$ be the index set of all tasks processed by agent i under the assignment x. Then $Q_x(i) = \{j | x_{ij} = 1\}$. If $\sum_{j\in Q_x(i)} a_{ijl} \le b_{il}$ for $i = 1, 2, \ldots, m, l = 1, 2, \ldots, u$, and if each task is assigned to exactly one agent, then x is called a *feasible assignment*. The family of all feasible assignments constitutes the set of feasible solutions of the MRGAP and we denote it by $\mathfrak{F}$.

Let $S \subseteq \{1, 2, \ldots, n\}$ and $S' = \{1, 2, \ldots, n\} \setminus S$. Let $\hat{x}$ be a given assignment. Define

$$F(S, \hat{x}) = \{x | x \in \mathfrak{F} \text{ and } x_{ij} = \hat{x}_{ij} \text{ for } j \in S, i \in M\}.$$

Thus $F(S, \hat{x})$ consists of all solutions of the MRGAP that 'agree' with $\hat{x}$ on positions corresponding to the columns given by S. If $|S'| = k$, then we call $F(S, \hat{x})$ an *S-restricted k-exchange neighborhood* of $\hat{x}$ or simply an *S-restricted neighborhood of* $\hat{x}$. S' is called the *ejection set*, and S is called the *binding set*.

For small values of $|S|$, searching $F(S, \hat{x})$ is almost equivalent to solving the MRGAP itself, and when $|S|$ is large, the neighborhood $F(S, \hat{x})$ becomes weak. Thus to develop a reasonable algorithm using the neighborhood, we need to keep a balance between these extreme cases. The best member in $F(S, \hat{x})$ can be obtained by solving the following MRGAP, which is a subproblem of the original MRGAP:

$$\begin{aligned} MRGAP(S,\hat{x}):\ \text{Minimize} \quad & \sum_{i=1}^{m}\sum_{j\in S'} c_{ij}x_{ij} \\ \text{Subject to} \quad & \sum_{j\in S'} a_{ijl}x_{ij} \le \hat{b}_{il}, \quad i = 1, 2, \ldots, m;\ l = 1, 2, \ldots, u \\ & \sum_{i=1}^{m} x_{ij} = 1, \quad j \in S' \\ & x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, m;\ j \in S' \end{aligned}$$

where $\hat{b}_{il} = b_{il} - \sum_{j\in Q_{\hat{x}}(i)\cap S} a_{ijl}$, for $i = 1, 2, \ldots, m$, $l = 1, 2, \ldots, u$, and summation over empty set is taken as zero. Any solution to MRGAP$(S, \hat{x})$ is called an *augmenting matrix* associated with the binding set S. Given an augmenting matrix $y = (y_{ij})$, a new solution $\bar{x} = (\bar{x}_{ij})_{m\times n}$ to the MRGAP can be obtained from $\hat{x}$ as

$$\bar{x}_{ij} = \begin{cases} \hat{x}_{ij}, & \text{if } i \in M, j \in S; \\ y_{ij}, & \text{if } i \in M, j \notin S. \end{cases} \tag{1}$$

By construction, $\bar{x}$ is feasible and we call it the *augmented solution*. For clarity, we sometimes use the label $A(y, \hat{x})$ for the augmented solution $\bar{x}$ obtained from y and $\hat{x}$.

For moderately large size of $|S'|$, the S-restricted exchange neighborhood $F(S,\hat{x})$ can be explored exactly with reasonable efficiency or can be searched approximately by means of an effective MRGAP heuristic. The neighborhood $F(S,\hat{x})$ can be viewed as a generalization of a neighborhood considered by [27] when elements of S are selected such that $|S' \cap Q_{\hat{x}}(i)| \le 1$ for all i; i.e. for each agent i, there is at most one task j for which $\hat{x}_{ij}$ is not fixed at value one.

Let $F_k(\hat{x}) = \cup_{S\subset N}\{F(S,\hat{x}) : |S| = n-k\}$. We call $F_k(\hat{x})$ the *k-exchange neighborhood* of $\hat{x}$ for the MRGAP. $F_k(\hat{x})$ is searched approximately by searching $F(S,\hat{x})$, for a suitably generated binding set S.

2.1. Heuristic – general framework

The neighborhood $F(S,\hat{x})$ is searched for improving solutions using a general-purpose *Integer Programming* (IP) code by solving MRGAP($S,\hat{x}$) to optimality or near optimality. The IP solver is used in a time-restricted manner. Recently, considerable interest has been observed in developing a heuristic for discrete optimization problems by making use of general-purpose IP solvers [28–31]. Our results are yet another contribution in this line of research that successfully uses variable fixing strategies.

A general outline of the heuristic is given in Algorithm 1. Here, $\hat{x}$ is the current solution of the original problem, x^* is the best solution found so far, y is a solution to MRGAP($S,\hat{x}$) generated by the IP solver, and $A(y,\hat{x})$ is the *augmented solution* generated by combining y and $\hat{x}$ as given by Eq. (1). Let $z(x)$ and $z(A(y,\hat{x}))$ denote the objective function values of the solution x and of the augmented solution $A(y,\hat{x})$, respectively.

```
Algorithm 1
Input: MRGAP, C
begin
    generate feasible solution $\hat{x}$
    $x^* \leftarrow \hat{x}$
    while (stopping criterion C is not satisfied)
        generate a binding set S using appropriate rules
        solve MRGAP($S,\hat{x}$) by IP solver, and let y be the solution produced
        compute the augmented solution $A(y,\hat{x})$
        if ($z(A(y,\hat{x})) < z(\hat{x})$)
            $\hat{x} \leftarrow A(y,\hat{x})$
            if ($z(\hat{x}) < z(x^*)$)
                $x^* \leftarrow \hat{x}$
            end if
        end if
    end while
    return $x^*$
end
```

Note that the neighborhood $F(S,\hat{x})$ could often be a very large-scale neighborhood (VLSN) for appropriately chosen S. We use the general-purpose IP solver CPLEX for searching $F(S,\hat{x})$, i.e. for solving the MRGAP($S,\hat{x}$). The algorithm also takes advantages of a possibility for providing a good (partial) solution as input to CPLEX.

2.2. Choice of the binding set

The selection of the binding set S (tasks that are going to be fixed to a particular agent) is crucial to performance of our algorithm. Clever choices of S could explore large regions of $F_k(\hat{x})$ by simply exploring $F(S,\hat{x})$. We now introduce nine greedy type heuristics for choosing the binding set S. Each selection of S defines a neighborhood $F(S,\hat{x})$. Note that $Q_{\hat{x}}(i)$ is the collection of column indices (tasks) that are assigned to agent i under $\hat{x}$.

Let $\Omega_i(\hat{x}) = (\tau_1,\tau_2,\ldots,\tau_{q_i})$ be an ordering of elements of $Q_{\hat{x}}(i)$ such that $\frac{c_{1\tau_1}}{\sum_l a_{1\tau_1 l}} \le \frac{c_{2\tau_2}}{\sum_l a_{2\tau_2 l}} \le \cdots \le \frac{c_{q_i\tau_{q_i}}}{\sum_l a_{q_i\tau_{q_i} l}}$ and $q_i = |Q_{\hat{x}}(i)|$.

Also, let $X = \{(i,j) : \hat{x}_{ij} = 1\}$ and $\Omega^X(\hat{x}) = ((i_1,j_1),(i_2,j_2),\ldots,(i_n,j_n))$ be an ordering of elements of X such that $\frac{c_{i_1j_1}}{\sum_l a_{i_1j_1l}} \le \frac{c_{i_2j_2}}{\sum_l a_{i_2j_2l}} \le \cdots \le \frac{c_{i_nj_n}}{\sum_l a_{i_nj_nl}}$. We denote the ordering $(j_1,j_2,\ldots,j_n)$ of tasks by $\Omega(\hat{x})$.

Let $|S'| = k$ and thus $|S| = n-k$. Introduce the parameters α_i, $1 \le i \le m$ such that $\sum_{i=1}^m \alpha_i = n-k$, and let $\beta_i = \lfloor \frac{\alpha_i}{2} \rfloor$. In our experiments we have selected the values of α_i such that they are approximately equal to $\min\{\lfloor \frac{n-k}{m} \rfloor, |Q_{\hat{x}}(i)|\}$.

We propose the following rules for choosing the binding set S taking into consideration the corresponding cost and resource values in various forms and combinations:

(R_1) Let $S_i = \{\tau_1,\tau_2,\ldots,\tau_{\alpha_i}\}$ and set $S = \bigcup_{i=1}^m S_i$.
(R_2) Let $T_i = \{\tau_{q_i},\tau_{q_i-1},\ldots,\tau_{q_i-\alpha_i+1}\}$ and set $S = \bigcup_{i=1}^m T_i$.
(R_3) Let $U_i = \{\tau_1,\tau_2,\ldots,\tau_{\beta_i}\} \cup \{\tau_{q_i},\tau_{q_i-1},\ldots,\tau_{q_i-\beta_i+1}\}$ and $S = \bigcup_{i=1}^m U_i$.

(R_4) $S = \{\bigcup_{i=1}^{p} S_i\} \bigcup \{\bigcup_{i=p+1}^{m} T_i\}$ where $p = \lfloor m/2 \rfloor$.
(R_5) $S = \{j_1, j_2, \ldots, j_{n-k}\}$.
(R_6) $S = \{j_n, j_{n-1}, \ldots, j_{n-k+1}\}$.
(R_7) $S = \{j_1, j_2, \ldots, j_r\} \bigcup \{j_n, j_{n-1} \ldots, j_{n-r}\}$ where $r = \lfloor (n-k)/2 \rfloor$.
(R_8) In this rule, we select S in a 'controlled' random way. Select approximately $(n-k)/10$ elements from the first 10% of elements of $\Omega(\hat{x})$, select approximately $(n-k)/10$ elements from the second 10% of elements of $\Omega(\hat{x})$, and continue this process so that $n-k$ elements from $\Omega(\hat{x})$ are selected and S is the resulting collection.
(R_9) Meta-neighborhood: generate $\lceil \frac{n}{n-k} \rceil$ different binding sets S by selecting $(n-k)$ consecutive elements from the ordering $\Omega(\hat{x})$ discussed above. For example, if $n = 100$ and $|S| = 20$, the following five versions of the binding set S are generated: fixing the first 20 assignments, the second 20, the third 20, the fourth 20, and the fifth 20.

For implementation purposes, we need not create the ordering $\Omega^X(x)$ or $\Omega(x)$ for each feasible solution x. We can simply order all the cells (i, j), $1 \le i \le m$; $1 \le j \le n$ globally once as per the cost/(sum of capacities) ratio, and the required ordering $\Omega^X(x)$ can easily be extracted if and when needed. Likewise, we need not create the ordering $\Omega_i(x) = \{\tau_1, \tau_2, \ldots, \tau_{q_i}\}$ for each solution x. We could simply construct the ordering of the cells (i, j), $1 \le j \le n$ as per the cost/(sum of capacities) ratio for each $i = 1, 2, \ldots, m$, and the required ordering can be easily extracted.

2.3. Starting solution

To initiate the algorithm, we need a starting solution. Note that computing a feasible solution to the MRGAP is NP-hard. However, modifying the problem slightly by introducing a dummy agent $m+1$ with $c_{m+1,j}$ equal to a large number, $a_{m+1,j,l} = 1$, and $b_{m+1,l} = n$, a starting feasible solution can easily be obtained. We call this new problem a *fictitious MRGAP*. Note that if the original MRGAP is feasible, an optimal solution to the fictitious MRGAP is also an optimal solution to the original MRGAP. Thus, we could work with the fictitious MRGAP instead of the original MRGAP. The greedy heuristics applied on the fictitious MRGAP is similar to the greedy heuristic for finding an initial GAP solution as reported in [26]. Straightforward implementation of this greedy algorithm takes $O(m^2n^2)$ time but it can be implemented in $O(mn \log(mn))$ time. We omit the details on computing the initial solution. An interested reader may refer to [26].

2.4. Detailed algorithm and parameter settings

Our *very large-scale variable neighborhood search* or *VLSVN search* is discussed in detail in this section along with various parameter settings identified by preliminary experiments.

We observed that if CPLEX was given a high-quality feasible solution as its starting solution, CPLEX required shorter computational time. If the starting solution is very close to optimal, CPLEX often terminates with an optimal solution almost instantly. This observation was crucial in the design and parameter settings of our algorithm since we explore the neighborhoods using CPLEX in a time-restricted manner. Initially the size of the ejection set S' is taken to be small so that the resulting MRGAP$(S, \hat{x})$ is also small, so that CPLEX can solve it efficiently and might produce an improved solution. Using this improved solution as the new starting solution, the size of the ejection set can be increased (i.e. the size of the binding set is decreased), gradually resulting in stronger neighborhoods but with the advantage that a good starting solution can be supplied to CPLEX in solving the resulting larger MRGAP$(S, \hat{x})$. We also gradually increase the time limit for CPLEX to perform a more aggressive search of the neighborhood.

This strategy of expanding the neighborhood size gradually and increasing the time limit for CPLEX can be viewed as a gradual *intensification* step. Thus, our local search algorithm is an intensified local search. The vector $\vec{p} = (p_1, p_2, \ldots, p_h)$ containing the sizes of the binding sets in various iterations is called the *binding size vector*. The corresponding time limits for CPLEX recorded in a vector $\vec{t} = (t_1, t_2, \ldots, t_h)$ is called the *time limit vector*.

The stopping criterion for the algorithm is the pre-specified total time limit T. A binding set S is chosen according to the rules in the order given in Section 2.2. Note that we have $8 + \lceil \frac{n}{n-k} \rceil$ different types of binding set selection rules. A solution in the neighborhood $F(S, \hat{x})$ is selected by running CPLEX for a prescribed amount of time to solve MRGAP$(S, \hat{x})$. The algorithm moves to this solution if it is better than the current solution. If no improvement is achieved for q number of consecutive iterations, the size of the neighborhood is increased and the duration of the neighborhood search time limit is extended, according to the binding size vector and the time limit vector. Here, q is a pre-specified parameter. When we reach the end of the intensification schedule, a local search is continued with the last schedule elements $\{p_h, t_h\}$ until the stopping criterion is met.

Note that if enough time is available, a diversification step may be added to the algorithm. We skip this extension in this paper due to the short time limits we used in our experimental study.

We describe the algorithm VLSVN search in the pseudocode below. The definitions of the parameters are as follows: MRGAP is the problem to be solved, T is the time limit for the algorithm, G is the total number of rules used for generating the binding sets, x is the current solution, x^* is the best solution found so far, y is the solution to MRGAP(S, x), and $A(y, S)$ is the *augmented solution*. The parameter q indicates when an intensification step will be called upon to change the size of the neighborhood and the duration of the neighborhood search.

Table 1
Intensification vectors: $|S|$ and CPLEX time limits.

Schedule	
*Sch*1	$\vec{p} = (60 \quad 40 \quad 20 \quad 10 \quad 5 \quad 5)$ $\vec{t} = (5 \quad 10 \quad 20 \quad 40 \quad 80 \quad 160)$
*Sch*2	$\vec{p} = (40 \quad 10 \quad 5)$ $\vec{t} = (10 \quad 40 \quad 80)$

Algorithm VLSVN Search

```
Input: MRGAP, $T$, $G$, $\vec{p}$, $\vec{t}$, $q$
  begin
    generate an initial feasible solution $x$
    $x^* \leftarrow x$
    $e \leftarrow 1$
    $p \leftarrow p_e$
    $t \leftarrow t_e$
    $g \leftarrow 0$
    while (running time $< T$)
        generate a binding set $S$ using rule $R_g$
        solve MRGAP$(S, x)$ by CPLEX within time $t$, and let $y$ be the solution produced
        compute the augmented solution $A(y, x)$
        if ($z(A(y, x)) < z(x)$)
          $x \leftarrow A(y, x)$
          if ($z(x) < z(x^*)$ )
            $x^* \leftarrow x$
          end if
        end if
        if (($x$ has not been improved in the last $q$ iterations) and ($e < h$))
          $e \leftarrow e + 1$
          $p \leftarrow p_e$
          $t \leftarrow t_e$
        end if
        $g \leftarrow (g + 1) \bmod G$
    end while
    return $x^*$
  end
```

3. Experimental study

The VLSVN search algorithm was coded in C++ and tested on a Dell workstation with an Intel Xeon 2.0 GHz processor, 512 MB memory, GNU g++ compiler version 3.2, and Linux (Mandrake 9.2) operating system. To explore the neighborhoods we have used CPLEX 9.1 with Concert Technology.

As a test bed we used problem instances generated by [21]. The instances are generated from the standard benchmark GAP instances of types C, D and E with 100 and 200 tasks. The GAP problem instances of type C and D are generated by J.E. Beasley and are part of the OR-Library [22]. All other instances are generated by [15].

The stopping criteria for our heuristic are the same as the time limits used in [21]: instances with 100 tasks were solved with the time limit of 300 s, and instances with 200 tasks were solved with the time limit of 600 s.

The size of the binding set S (the number of task assignments to be fixed) has been expressed as a percentage of the total number of tasks. The binding size vector $\vec{p}$ contains these percentages. The time limit vector $\vec{t}$ contains the time limit in CPU seconds for CPLEX to solve MRGAP(S, x). Table 1 summarizes the two intensification schedules used in our experiments. The results are reported for the two schedules in order to show that both are successful, regardless of the fact that there is a difference in their length.

The CPLEX parameters are set as follows: MIPEmphasis $= 4$ (hidden feasibility) [32], TiLim $= t$ (in seconds) as given by components of the vector $\vec{t}$, and MIPStart $= 1$ (CPLEX is provided with an initial feasible solution).

The initial starting solution is obtained as discussed in Section 2.3. The starting solution for CPLEX when solving MRGAP(S, x) is generated by restricting the current solution x using the binding set S.

Computational results are summarized in Tables 2–4. The solutions obtained by our heuristic (VLSVN search) are compared to the solutions obtained by the tabu search algorithm (TS(Yag)) proposed by [21], and by a straightforward application of CPLEX with MIPEmphasis $= 4$ (hidden feasibility). Since [21] reported on three versions of the tabu search

Table 2
Instances of type C. Comparison with CPLEX (MIP solver) and the tabu search method by [21]. Algorithm VLSVNS and CPLEX were run on a Dell workstation with an Intel Xeon 2.0 GHz processor with time limits of 300 and 600 s for $n = 100$ and 200, respectively. The tabu search was run on a Sun Ultra 2 Model 2300 with the same time limits. Columns labeled with *TTB* contain the time when the best solution is reached.

Type	*n*	*m*	*u*	*LB*	CPLEX			TS (Yag)		VLSVNS (Sch1)		VLSVNS (Sch2)	
					Cost	*TTB*	Time	Cost	*TTB*	Cost	*TTB*	Cost	*TTB*
C	100	5	1	†1931	*1931	<1	0.65	*1931	1.25	*1931	0.64	*1931	0.64
C	100	5	2	†1933	*1933	<1	0.86	*1933	2.39	*1933	0.86	*1933	0.87
C	100	5	4	†1943	*1943	<3	9.56	*1943	172.35	*1943	5.14	*1943	9.56
C	100	5	8	†1950	*1950	34	86.51	*1950	61.19	*1950	22.57	*1950	56.71
C	100	10	1	†1402	*1402	4	6.39	*1402	5.49	*1402	8.41	*1402	6.43
C	100	10	2	†1409	*1409	9	9.97	*1409	133.34	*1409	8.3	*1409	10.09
C	100	10	4	†1419	*1419	5	6.12	*1419	37.40	*1419	5.14	*1419	6.1
C	100	10	8	†1435	*1435	50	283.3	*1435	147.34	*1435	24.56	*1435	145.28
C	100	20	1	†1243	*1243	<1.77	1.77	1245	55.48	*1243	1.82	*1243	1.8
C	100	20	2	†1250	*1250	3	3.6	1251	45.39	*1250	3.62	*1250	3.6
C	100	20	4	†1254	*1254	12	12.44	1256	186.09	*1254	12.18	*1254	16.9
C	100	20	8	†1267	*1267	10	155.1	*1267	205.60	*1267	5.25	*1267	10.64
C	200	5	1	†3456	*3456	<1	1.18	*3456	170.10	*3456	1.21	*3456	1.2
C	200	5	2	†3461	*3461	17	49.19	*3461	47.64	*3461	17.3	*3461	23.68
C	200	5	4	†3466	*3466	2	77.24	*3466	167.53	*3466	5.25	*3466	10.31
C	200	5	8	†3473	*3473	102	248.05	*3473	530.30	*3473	34.91	*3473	183.06
C	200	10	1	†2806	*2806	88	110.32	2807	46.23	*2806	18.9	*2806	124.21
C	200	10	2	†2811	*2811	188	331.13	2812	316.68	*2811	156.56	*2811	54.02
C	200	10	4	†2819	*2819	6	374.64	2821	399.99	*2819	112.81	*2819	10.29
C	200	10	8	2833	*2837	5	–	*2837	344.24	*2837	5.44	*2837	10.41
C	200	20	1	†2391	*2391	37	50.16	2393	369.54	*2391	182.21	*2391	60.9
C	200	20	2	†2397	*2397	14	183.77	2398	313.10	*2397	16.01	*2397	61.52
C	200	20	4	2408	*2409	36	–	*2409	430.56	*2409	245.36	*2409	261.17
C	200	20	8	2415	*2417	490	–	2422	47.29	*2417	77.05	*2417	309.55

Table 3
Instances of type D. Comparison with CPLEX (MIP solver) and the tabu search method by [21]. Algorithm VLSVNS and CPLEX were run on a Dell workstation with an Intel Xeon 2.0 GHz processor with time limits of 300 and 600 s for $n = 100$ and 200, respectively. The tabu search was run on a Sun Ultra 2 Model 2300 with the same time limits. Columns labeled with *TTB* contain the time when the best solution is reached.

Type	*n*	*m*	*u*	*LB*	CPLEX			TS (Yag)		VLSVNS (Sch1)		VLSVNS (Sch2)	
					Cost	*TTB*	Time	Cost	*TTB*	Cost	*TTB*	Cost	*TTB*
D	100	5	1	† 6353	6354	107	–	6357	109.87	6356	221.35	* 6353	111.95
D	100	5	2	6352	*6355	87	–	6359	136.10	6358	21.78	6357	112.37
D	100	5	4	6362	*6370	208	–	6379	207.25	6377	174.2	6373	71.44
D	100	5	8	6388	6420	193	–	6425	67.89	6417	173.57	*6411	112.84
D	100	10	1	6342	*6359	254	–	6361	246.00	6360	66.1	*6359	243.96
D	100	10	2	6340	*6367	54	–	6378	174.39	6370	168.65	*6367	243.59
D	100	10	4	6361	6421	204	–	6430	274.92	*6416	178.11	6428	152.79
D	100	10	8	6388	6476	25	–	6478	241.80	*6474	203.72	6477	20.48
D	100	20	1	6177	*6216	155	–	6231	194.94	6232	228.46	6217	285.64
D	100	20	2	6165	6249	229	–	6261	253.83	6241	60.79	*6228	122.82
D	100	20	4	6182	6271	232	–	6321	277.59	*6270	241.65	6286	112.51
D	100	20	8	6206	6436	100	–	6481	234.49	6453	228.59	*6429	243.84
D	200	5	1	12741	*12744	228	–	12751	191.63	12745	107.5	12745	30.83
D	200	5	2	12751	12759	66	–	12766	441.53	12760	245.48	*12756	132.95
D	200	5	4	12745	12754	163	–	12775	178.92	12754	189.39	*12749	601.19
D	200	5	8	12755	12781	580	–	12805	527.78	*12777	417.47	12778	605.8
D	200	10	1	12426	12440	538	–	12463	330.52	*12439	529.34	12447	203.86
D	200	10	2	12431	*12455	358	–	12477	476.07	12459	326.57	12459	601.48
D	200	10	4	12432	*12460	135	–	12496	471.01	12462	485.3	12466	448.93
D	200	10	8	12448	12532	275	–	12571	481.10	12527	590.77	*12522	600.72
D	200	20	1	12230	12273	96	–	12312	230.72	*12271	357.2	12277	602.33
D	200	20	2	12227	*12290	554	–	12332	597.11	*12290	500.25	12291	507.88
D	200	20	4	12237	*12319	330	–	12396	576.45	12321	112.36	12349	498.2
D	200	20	8	12254	12448	80	–	12485	337.27	12460	512.35	*12424	599.49

heuristic, in our tables we list the best solutions achieved by these three versions. For our algorithm we report the outcomes for the two different intensification schedules in separate columns.

The best solutions are marked by *. The time in seconds to reach a solution is reported in the column labeled *TTB* (time-to-best). The column labeled by *LB* contains lower bounds for the problem instances. The *LB* equal to the optimal value is marked by †. The CPLEX columns contain the time-to-best (*TTB*) and the running times of CPLEX (*time*). These time-to-best

Table 4
Instances of type E. Comparison with CPLEX (MIP solver) and the tabu search method by [21]. Algorithm VLSVNS and CPLEX were run on a Dell workstation with an Intel Xeon 2.0 GHz processor with time limits of 300 and 600 s for $n = 100$ and 200, respectively. The tabu search was run on a Sun Ultra 2 Model 2300 with the same time limits. Columns labeled with *TTB* contain the time when the best solution is reached.

Type	*n*	*m*	*u*	*LB*	CPLEX			TS (Yag)		VLSVNS (Sch1)		VLSVNS (Sch2)	
					Cost	*TTB*	Time	Cost	*TTB*	Cost	*TTB*	Cost	*TTB*
E	100	5	1	†12681	*12681	9	22.09	*12681	54.08	*12681	15.02	*12681	27.4
E	100	5	2	†12692	*12692	1	47.96	*12692	120.93	*12692	5.11	*12692	10.08
E	100	5	4	†12810	*12810	7	–	12812	104.03	*12810	133.5	*12810	10.18
E	100	5	8	†12738	*12738	88	–	*12738	53.98	*12738	220.89	*12738	20.84
E	100	10	1	†11577	*11577	147	193.02	*11577	90.03	*11577	35.5	*11577	228.4
E	100	10	2	†11582	11587	51	–	11587	179.65	11587	184.47	*11582	261.3
E	100	10	4	11636	11673	262	–	11676	289.32	*11672	298.83	11684	192.37
E	100	10	8	11619	*11657	184	–	11701	260.90	11665	122.33	11693	111.74
E	100	20	1	† 8436	8446	208	–	8447	142.39	* 8445	143.57	8450	97.92
E	100	20	2	10123	10157	280	–	10150	207.09	*10149	263.96	10163	283.81
E	100	20	4	10794	11092	110	–	*11029	160.57	11070	255.17	11097	293.16
E	100	20	8	11224	11806	257	–	*11610	265.33	11824	294.12	11685	272.9
E	200	5	1	†24930	24931	8	9.15	24933	43.21	*24930	7.54	24931	9.17
E	200	5	2	†24933	*24933	12	12.31	24936	430.29	*24933	8.19	*24933	13.04
E	200	5	4	24990	*24991	302	456.44	24999	537.27	*24991	252.06	*24991	233.42
E	200	5	8	†24943	*24943	8.5	8.8	24950	192.28	*24943	26.61	*24943	8.9
E	200	10	1	†23307	*23307	107	130.74	23312	411.12	*23307	76.2	*23307	103.43
E	200	10	2	23310	*23312	12	131.36	23317	436.01	*23312	86.91	*23312	104.58
E	200	10	4	23344	23361	463	–	23363	376.65	23365	506.35	*23360	486.84
E	200	10	8	23339	23373	422	–	23410	198.96	*23376	472.77	23386	91.74
E	200	20	1	†22379	*22379	360	–	22386	178.16	*22379	490.36	*22379	243.48
E	200	20	2	22387	22406	450	–	22408	333.97	*22405	518.12	22415	141.7
E	200	20	4	22395	22450	428	–	*22439	462.53	22449	467.6	*22439	132.36
E	200	20	8	22476	22701	510	–	*22614	317.52	22793	453.93	22631	558.84

values are approximate because the time-to-best is not explicitly available from CPLEX. The *time* is reported for the instances solved to optimality by CPLEX within the pre-specified time limits. Otherwise, this column contains '—'.

The solutions achieved by our heuristic are better or equal in quality compared to the solutions reported in the literature, with a few exceptions. For these instances [21] reported better results or CPLEX achieved better solutions.

For the C instances, the best solutions were achieved by CPLEX and by our heuristic (VLSVNS) for all 24 instances. For the D instances, the best solutions were achieved by CPLEX in 10 cases, by VLVNS with intensification schedule *Sch*1 in 7 cases and by VLVNS with intensification schedule *Sch*2 in 10 cases. Unique best solutions were achieved by CPLEX for 7 instances, by VLVNS (*Sch*1) for 6 instances and by VLVNS (*Sch*1) for 8 instances. For the E instances, the best solutions were achieved by CPLEX in 12 cases, by the tabu search TS(Yag) in 8 cases, by VLVNS (*Sch*1) in 17 cases and by VLVNS (*Sch*2) in 14 cases. Unique best solutions were achieved by CPLEX for 1 instance, by TS(Yag) for 2 instances, by VLVNS (*Sch*1) for 6 instances and by VLVNS (*Sch*1) for 3 instances.

4. Conclusion

In this paper we introduced an efficient local search heuristic for MRGAP. The intensified local search is a VLSVN search since it combines ideas of VLSN search and VN search. The general-purpose IP solver CPLEX is used in a novel way to explore the underlying very large-scale neighborhoods. We have obtained improved solutions for several benchmark problems available in the literature with reasonable computational effort. Our algorithms are particularly of interest when good-quality solutions need to be obtained quickly. Since our local search algorithm may be embedded in any metaheuristic, we believe that a combination with the tabu search algorithm of [21] may result in a superior algorithm. The ideas used in the paper are applicable to solving several other combinatorial optimization problems.

Acknowledgement

We are thankful to a referee for his/her comments which improved the presentation.
This work is partially supported by an NSERC grant awarded to Abraham P. Punnen.

References

[1] B. Gavish, H. Prikul, Allocation of databases and processors in a distributed computing system, in: J. Akoka (Ed.), Management of Distributed Data Processing, North-Holland Publishing Company, Amsterdam, 1982.
[2] B. Gavish, H. Prikul, Computer and database location in distributed computer systems, IEEE Transactions on Computers 35 (1986) 583–590.
[3] R.A. Murphy, A private fleet model with multi-stop backhaul, Working paper 103, Optimal Decision Systems, Green Bay, WI54306, 1986.
[4] H. Pirkul, An integer programming model for allocation of databases in a distributed computer system, European Journal of Operational Research 26 (1986) 401–411.
[5] D. Cattrysse, L.N. Van Wassenhove, A survey of algorithms for the generalized assignment problem, European Journal of Operational Research 60 (1992) 260–272.

[6] I.H. Osman, Heuristics for the generalized assignment problem: Simulated annealing and tabu search approaches, OR Spektrum 17 (1995) 211–225.
[7] R.M. Nauss, Solving the generalized assignment problem: An optimizing and heuristic approach, INFORMS Journal on Computing 15 (2003) 249–266.
[8] J.S. Park, B.H. Lim, Y. Lee, A Lagrangian dual-based branch-and-bound algorithm for the generalized assignment problems, Management Science 24 (1998) 345–357.
[9] M. Savelsbergh, A branch-and-price algorithm for the generalized assignment problem, Operations Research 45 (1997) 831–841.
[10] S. Martello, P. Toth, An algorithm for the generalized assignment problem, in: J.P. Brans (Ed.), Operational Research '81, North-Holland, 1981, pp. 589–603.
[11] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, New York, 1990.
[12] P.C. Chu, J.E. Beasley, A genetic algorithm for the generalized assignment problem, Computers and Operations Research 24 (1997) 17–23.
[13] M. Laguna, J.P. Kelly, J.L. González-Velarde, F. Glover, Tabu search for the multilevel generalized assignment problem, European Journal of Operational Research 82 (1995) 176–189.
[14] H.R. Lourenço, D. Serra, 'Adaptive approach heuristic for the generalized assignment problem, Technical report: Department of Economics and Management, Universitat Pompeu Fabra, R. Trias Fargas 25–27, 08005 Barcelona, Spain, 1998. Available at: http://www.econ.upf.es/deehome/what/wpapers/listwork.html.
[15] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem, INFORMS Journal on Computing 16 (2004) 133–151.
[16] L.A.N. Lorena, M.G. Narciso, Relaxation heuristic for the generalized assignment problem, European Journal of Operational Research 91 (1996) 600–610.
[17] M. Trick, A linear relaxation heuristic for the generalized assignment problem, Naval Research Logistic 39 (1992) 137–151.
[18] D. Cattrysse, M. Salomon, L.N. Van Wassenhove, A set partitioning heuristic for the generalized assignment problem, European Journal of Operational Research 72 (1994) 167–174.
[19] M. Guignard, S. Zhu, A two phase dual algorithm for solving Lagrangian duals in mixed integer programming, Working paper, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1996.
[20] B. Gavish, H. Prikul, Algorithms for the multi-resource generalized assignment problem, Management Science 37 (1991) 695–713.
[21] M. Yagiura, S. Iwasaki, T. Ibaraki, F. Glover, A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem, Discrete Optimization 1 (1) (2004) 87–98.
[22] OR-Library available at http://people.brunel.ac.uk/mastjjb/jeb/info.html.
[23] R.K. Ahuja, O. Ergun, A. Punnen, Very large scale neighborhood search: Theory, algorithms, and applications, in: T. Gonzalez (Ed.), Handbook of Approximation Algorithms and Metaheuristics, in: Computer and Information Science Series, vol. 10, Chapmann and Hall, CRC Press, 2006.
[24] R.K. Ahuja, O. Ergun, A. Punnen, A survey of very large scale neighborhood search techniques, Discrete Applied Mathematics 23 (2002) 75–102.
[25] N. Mladenović, P. Hansen, Variable neighborhood search, Computers and Operations Research 24 (1997) 1097–1100.
[26] S. Mitrovic-Minic, A.P. Punnen, Very large-scale variable neighborhood search for the generalized assignment problem, Journal of Interdisciplinary Mathematics 11 (2008) 653–670.
[27] T. Oncan, S.N. Kabadi, K.P.N. Nair, A.P. Punnen, VLSN search algorithm for partitioning problems using matching neighborhoods, Journal of the operational research society 59 (2008) 388–398.
[28] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighbor- hoods to improve MIP solutions, Mathematical Programming 102 (2005) 71–90.
[29] M. Fischetti, A. Lodi, Local branching, Mathematical Programming 98 (2003) 23–47.
[30] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research 34 (2007) 2403–2435.
[31] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (4) (2006) 455–472.
[32] ILOG ILOG cplex 9.0 User's Manual, 2003.
[33] M. Yagiura, T. Ibaraki, F. Glover, A path relinking approach with ejection chains for the generalized assignment problem, European Journal of Operational Research 169 (2006) 548–569.