

Solving the clique partitioning problem as a maximally diverse grouping problem

Jack Brimberg · Stefana Janićijević ·
Nenad Mladenović · Dragan Urošević

Received: 4 March 2014 / Accepted: 16 February 2015 / Published online: 4 March 2015
© Springer-Verlag Berlin Heidelberg 2015

Abstract In this paper we show that the clique partitioning problem can be reformulated in an equivalent form as the maximally diverse grouping problem (MDGP). We then modify a *skewed general variable neighborhood search* (SGVNS) heuristic that was first developed to solve the MDGP. Similarly as with the MDGP, significant improvements over the state of the art are obtained when SGVNS is tested on large scale instances. This further confirms the usefulness of a combined approach of diversification afforded with *skewed VNS* and intensification afforded with the local search in *general VNS*.

Keywords Clique partitioning · Maximum diverse grouping problem · Variable neighborhood search

1 Introduction

The clique partitioning problem can be described as follows. Given a weighted, non-oriented and complete graph $G = (V, E, w)$, we wish to find a partition of the node

J. Brimberg
Royal Military College of Canada, Kingston, Canada
e-mail: Jack.Brimberg@rmc.ca

S. Janićijević · D. Urošević (✉)
Mathematical Institute SANU, Belgrade, Serbia
e-mail: draganu@mi.sanu.ac.rs

S. Janićijević
e-mail: stefana@mi.sanu.ac.rs

N. Mladenović
LAMIH, Université de Valenciennes, Valenciennes, France
e-mail: Nenad.Mladenovic@univ-valenciennes.fr

set into an unknown number of nonempty, disjoint subsets $V_1, V_2, \dots, V_k$ such that the sum of edge weights over the k induced subgraphs (or cliques) $G_1, G_2, \dots, G_k$ is minimized. Let c_{ij} denote the weight (or cost) of edge (i, j) , for all pairs of nodes (i, j) , $1 \leq i < j \leq n$, where n equals the number of nodes in G ($n = |V|$). A mathematical formulation of the problem is given by (see [2] or [12]):

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & x_{ij} + x_{jr} - x_{ir} \leq 1, \quad \forall 1 \leq i < j < r \leq n \quad (\text{CPP}) \\ & x_{ij} - x_{jr} + x_{ir} \leq 1, \quad \forall 1 \leq i < j < r \leq n \\ & -x_{ij} + x_{jr} + x_{ir} \leq 1, \quad \forall 1 \leq i < j < r \leq n \\ & x_{ij} \in \{0, 1\}, \quad \forall 1 \leq i < j \leq n. \end{aligned}$$

If $x_{ij} = 1$, then nodes i and j are in the same cluster, and the cost c_{ij} of edge (i, j) is added in the objective function; otherwise it is not. The constraint set ensures that all edges in the complete subgraph G_t , $t = 1, 2, \dots, k$, are included in the solution.

The clique partitioning problem (CPP) may be extended to incomplete graphs by inserting fictitious edges with large positive weights between pairs of nodes wherever edges are missing in the original graph. The large positive weight ensures that each subgraph G_t , $t = 1, 2, \dots, k$, is a clique that contains no fictitious edges. If all edges in a complete graph G have negative (or zero) weights, the problem becomes trivial, with the optimal solution given by $k = 1$ and $G_1 = G$. Thus, the CPP becomes interesting only when some of the edge weights have positive values.

The CPP has important applications in the social sciences (e.g., see [2], and the references therein). Wang et al. [16] demonstrate that the CPP compares favorably with K -means and latent class analysis for recovery of cluster structure in real data sets. An advantage of the CPP model is that we do not need to know the number of clusters beforehand. One of the most cited applications of CPP is in the aggregation of binary equivalence relations. In this context, the c_{ij} parameters represent the number of attributes on which nodes i and j disagree minus the number on which they agree. For example, if there are 10 attributes in all being compared and given nodes i and j have the same measurement on 8 of them, $c_{ij} = 2 - 8 = -6$, and there is a strong tendency to place these two nodes in the same cluster. If on the other hand $c_{ij} = 8 - 2 = 6$, there would be a tendency to place them in separate clusters [2].

The earliest methods to solve CPP use a simple relocation procedure that interchanges nodes across clusters until a local optimum is reached (e.g., [13]). Marcotorchino and Michaud [10] expanded the search to include merging of clusters and exchanging cluster memberships for pairs of nodes. Simulated annealing and tabu search heuristics are proposed by De Amorim et al. [4], based on the simple relocation neighborhood above. However, this also includes the possibility of relocating a node to the empty set, thereby increasing the number of clusters by 1. Charon and Hudry [3] investigate various “noising” procedures that distort the data in order to allow uphill moves. Brusco and Köhn [2] develop a state-of-the-art “neighborhood search heuris-

tic”, following the ideas of the variable neighborhood search (VNS) metaheuristic (see [11]), where the shaking (or perturbation) step is carried out in a random fashion instead of the systematic approach of VNS. The local search employs the single node relocation neighborhood.

In the next section we show how the CPP may be reformulated as a maximally diverse grouping problem. This allows us to modify a skewed general variable neighborhood search that we recently developed (Brimberg et al. [1]) to solve the maximally diverse grouping problem and apply this approach to the CPP (see Sect. 3 for details). Computational results are reported in Sect. 4 on a wide range of problem instances. It is interesting that Brusco and Köhn [2] reported similar results with their neighborhood search heuristic on test instances from [3], but significantly better results than Charon and Hudry [3] on larger problem instances. We arrive at a similar finding relative to the more recent paper. Finally in Sect. 5 we offer some conclusions and ideas for further research.

2 Relation to the maximally diverse grouping problem

The *maximally diverse grouping problem* (MDGP) requires finding a partition of a given set of elements (or nodes) into a specified number of mutually disjoint subsets (or groups) in order to maximize the overall diversity between elements of the same group. One of the earliest applications of MDGP was in the formation of student workgroups [18]. For example, in MBA programs [5] it is very important to divide the class into diverse study groups in order to enhance the learning environment. Another application cited is the formation of peer review groups to evaluate research proposals [7]. Again, the objective is to form diverse groups in order to ensure that projects are evaluated from several different points of view. Other applications have been cited in the literature; e.g., see [9] for exam scheduling, [17] for very large scale integration (VLSI) design, and [8] for the storage of large programs onto paged memory.

In the MDGP the number of groups (or cliques) is fixed, and typically these groups must have the same size, or nearly the same size. However, in the CPP we don’t know the number of groups, and there are no limitations on group size. To get around this difficulty, we simply set the number of groups initially to an upper limit of n (the number of nodes in G), and allow some of these groups to be empty. No limitations on group size are imposed, so that the number of groups is effectively treated as a variable in the model formulation.

Now let $y_{ig} = 1$, if node i belongs to group g , and 0 otherwise. We reformulate CPP as the following equivalent quadratic binary integer program:

$$\begin{aligned} \max & - \sum_{g=1}^n \sum_{i=1}^{n-1} \sum_{j=i+1}^n c_{ij} y_{ig} y_{jg} \\ \text{s.t.} & \quad \quad \quad (\text{CPP}_e) \\ & \sum_{g=1}^n y_{ig} = 1, \quad i = 1, \dots, n \\ & y_{ig} \in \{0, 1\}, \quad i = 1, \dots, n, \quad g = 1, \dots, n \end{aligned}$$

This formulation is equivalent to the general MDGP except that (a) all constraints on group size have been deleted, and (b) the number of groups is no longer specified. Note that the nonlinear formulation in (CPP_e) allows a reduction in the number of constraints compared to the original linear formulation (CPP) from $O(n^3)$ to n . Meanwhile the number of binary variables remains the same in both formulations.

A similar 0/1 quadratic program as (CPP_e) may be found in Wang et al. [15] as an alternate formulation of the clique partitioning problem. However, in [15], a maximum number of cliques ($k_{\max} < n$) is assumed to be known, which is not necessary here. Also, those authors use the quadratic formulation to solve the group technology problem, whereas we are showing here that the CPP may be viewed as relaxed form of the MDGP.

By recasting the model in the form of a MDGP, we are now able to borrow any solution method for MDGP and apply it (after some small modifications) to solve our original CPP. We will take advantage of this fact by adapting a new skewed general variable neighborhood search heuristic that was applied with great success on the MDGP (see [1] and [14]). For completeness the details of the new heuristic as applied to the CPP are given in the next section. Computational results in Sect. 4 demonstrate that the new heuristic provides a powerful solution approach.

3 Implementation

3.1 Solution space of MDGP

The solution space consists of all possible divisions of elements into groups. The current solution is represented by vector x^c of length n ($x^c = (x_1^c, x_2^c, \dots, x_n^c)$) such that x_i^c is the label of the group containing element i ($i = 1, 2, \dots, n$). In order to speed up the local search, we also maintain matrix sc^c such that sc_{ig}^c is the sum of edge weights between element i and all elements assigned to the group g in the current solution:

$$sc_{ig}^c = \sum_{j=1,2,\dots,n; x_j^c=g} c_{ij}.$$

Note that for the current solution, matrix sc^c can be computed in $O(n^2)$.

The number of groups is initially set to n with one element in each group. But, as explained later, the number of groups ($g_{\max}$) is allowed to change during the solution process. To accomplish this, we always maintain an empty group in the list of groups.

3.2 VND local search

The local search is implemented as a variable neighborhood descent (VND), for which the following neighborhoods are designed: Insertion and Swap.

Neighborhood Insertion contains solutions obtained by moving only one element from its current group to another group. By using the previously described matrix

sc^c , it is possible to efficiently compute, for each feasible move, the change of the objective function value. Denote with x^n a “neighboring” solution obtained from a solution x^c by moving element i from its current group g_1 to a group g_2 . In this case, the sum of edge weights in all groups except the groups g_1 and g_2 , are unchanged. The element i is removed from the group g_1 , and because of that, the sum of edge weights in the group g_1 decreases by the sum of weights of edges between i and all other elements belonging to the group g_1 . Element i is inserted into group g_2 ; hence, the sum of edge weights increases by the sum of weights of all edges between i and elements belonging to the group g_2 . It is easy to conclude that the difference between objective function values for solutions x^c and x^n is

$$\Delta f = f(x^n) - f(x^c) = sc_{ig_2}^c - sc_{ig_1}^c.$$

Algorithm 1: Local search in Insertion neighborhood

```

1 Function LSIns( $x, sc, f$ );
2  $rez \leftarrow \text{false}$ ;
3 for  $v \leftarrow 1$  to  $n$  do
4   for  $g \leftarrow 1$  to  $g_{\max}$  do
5     if  $x_v \neq g$  then
6        $df \leftarrow sc_{v,g} - sc_{v,x_v}$ ;
7       if  $df < 0$  then
8         Update $sc(x, sc, v, g)$ ;
9          $x_v \leftarrow g$ ;
10         $f \leftarrow f + df$ ;
11         $rez \leftarrow \text{true}$ ;
12      end
13    end
14  end
15 end
16 return  $rez$ 

```

If the insertion step is performed, we change the current solution, and then it is necessary to update matrix sc^c . If the element i moves from the group g_1 to the group g_2 , then groups g_1 and g_2 are modified and values $sc_{jg_1}^c$ and $sc_{jg_2}^c$ must be updated in the following way:

$$sc_{jg_1}^c = sc_{jg_1}^c - c_{ji}$$

and

$$sc_{jg_2}^c = sc_{jg_2}^c + c_{ji}.$$

Since this updating must be performed for each element j , updating of matrix sc^c after performing an insertion move has complexity $O(n)$. On the other hand, the cardinality of the Insertion neighborhood is $O(g_{\max}n)$, where $g_{\max}$ is the current number of groups (including the empty group). Note that if a group becomes empty as the result of an insertion move, it is eliminated, the remaining groups are renumbered accordingly,

and $g_{\max}$ is reduced by 1. If the insertion move results in group $g_{\max}$ (the empty one) becoming a singleton, $g_{\max}$ is increased by 1.

Neighborhood Swap contains solutions obtained by swapping a single pair of elements belonging to different groups (elements exchange the group they are currently assigned to). Let an element i be in a group g_i and an element j in a group g_j in current solution x^c . Denote with x^n the solution obtained after moving element i into group g_j and element j into group g_i . Since the element i is removed from the group g_i , the weights of edges between element i and elements remaining in the group g_i do not contribute to the objective function value of the new solution. But, because the element i is inserted in the group g_j , weights of all edges between i and the elements belonging to the group g_j contribute to the objective function value of the new solution. Similar facts are true for the element j . So, we can finally calculate the difference between the objective values of the current and neighboring solutions:

$$\Delta f = f(x^n) - f(x^c) = (sc_{ig_j}^c - sc_{ig_i}^c) + (sc_{jg_i}^c - sc_{jg_j}^c) - 2c_{ij}.$$

It is obvious that the change in objective value for each solution from Neighborhood Swap is done in $O(1)$, while the cardinality of Swap is $O(n^2)$. After performing a Swap move, it is necessary to update the matrix sc^c , and the complexity of this update is $O(n)$ (we can consider Swap or 2-opt as two successive Insertions).

Algorithm 2: Local search in Swap neighborhood

```

1 Function LSSwap( $x, sc, f$ );
2  $rez \leftarrow \text{false}$ ;
3 for  $v \leftarrow 1$  to  $n - 1$  do
4   for  $u \leftarrow v + 1$  to  $n$  do
5     if  $x_v \neq x_u$  then
6        $df \leftarrow sc_{v,x_u} + sc_{u,x_v} - sc_{v,x_v} - sc_{u,x_u} - 2d_{v,u}$ ;
7       if  $df < 0$  then
8         Swap( $x, v, u$ );
9          $f \leftarrow f + df$ ;
10        UpdateSc( $x, sc, v, u$ );
11         $rez \leftarrow \text{true}$ 
12      end
13    end
14  end
15 end
16 return  $rez$ ;
  
```

In brief the VND works as follows (for more details see [1]). The search is first conducted in Neighborhood Insertion until a local minimum is obtained. The search then proceeds to Neighborhood Swap. If after one cycle (see Algorithm 2), at least one Swap move is made, the search resumes in Neighborhood Insertion, otherwise the VND stops.

3.3 Skewed general variable neighborhood search

Variable neighborhood search (VNS) is a well-known metaheuristic, or framework for building heuristics, whose basic idea is a systematic change of neighborhood structures during a search for a better solution (Mladenović and Hansen [11]). The inner loop of the basic VNS variant contains 3 steps: (a) shaking; (b) local search; (c) neighborhood change. A successful VNS variant, called General VNS, uses a mechanism of changing neighborhoods not only in the diversification or Shaking step, but also in the intensification or deterministic local search step (Hansen et al. [6]). Skewed VNS is another VNS variant that modifies the 'neighborhood change' step. A current solution is allowed to move to an inferior solution only if the latter is very far and of similar quality. Recently Brimberg et al. [1] show that the combination of both variants, which they call Skewed General VNS (SGVNS), works very well in solving the MDGP. For example in testing 540 instances from the literature, 530 best known solutions were improved. This remarkable result is the main motivation of this paper. The pseudo-code of SGVNS for solving CPP is given in Algorithm 3.

Algorithm 3: Skewed General Variable Neighborhood Search (SGVNS)

```

1 Procedure SGVNS( $\alpha, k_{min}, k_{max}, k_{step}$ );
2  $f^c \leftarrow \text{InitialSolution}(x^c, sc^c)$ ;
3 VND( $x^c, sc^c, f^c$ );
4  $(x^b, sc^b, f^b) \leftarrow (x^c, sc^c, f^c)$ ;
5  $k \leftarrow k_{min}$ ;
6 while Stopping condition is not met do
7    $(x^n, sc^n, f^n) \leftarrow (x^c, sc^c, f^c)$ ;
8   Shake( $k, x^n, sc^n, f^n$ );
9   VND( $x^n, sc^n, f^n$ );
10  if  $(f(x^n) - \alpha d(x^n, x^c)|f(x^c)| < f(x^c))$  and  $(f(x^n) - \alpha d(x^n, x^b)|f(x^b)| < f(x^b))$ 
11  then
12     $(x^c, sc^c, f^c) \leftarrow (x^n, sc^n, f^n)$ ;
13    if  $f(x^c) < f(x^b)$  then
14       $(x^b, sc^b, f^b) \leftarrow (x^c, sc^c, f^c)$ ;
15       $k \leftarrow k_{min}$ 
16    else
17       $k \leftarrow k + k_{step}$ ;
18      if  $k > k_{max}$  then  $k \leftarrow k_{min}$  end
19    end
20  else
21     $k \leftarrow k + k_{step}$ ;
22    if  $k > k_{max}$  then  $k \leftarrow k_{min}$  end
23  end

```

Let us define $d(x^n, x^b)$, the distance between solutions x^n and x^b , in the following way:

$$d(x^n, x^b) = \frac{\left| \{(i, j) | 1 \leq i < j \leq n, ((x_i^b = x_j^b) \wedge (x_i^n \neq x_j^n)) \vee ((x_i^b \neq x_j^b) \wedge (x_i^n = x_j^n))\} \right|}{\sum_{g=1}^{g_b} \binom{c_g^b}{2}},$$

where g_b is the number of groups in the best solution x^b , and c_g^b is the number of elements in group g in the best solution.

Intuitively, the expression in the numerator is the number of pairs of elements belonging to the same cluster (or group) in one solution, but not belonging to the same group in the other solution. On the other hand the denominator is equal to the number of edges participating in cliques in the optimal solution. The expression $d(x^c, x^n)$ denotes the distance between solutions x^c and x^n and calculates in a similar way. The pseudo-code for the Shaking step is given in Algorithm 4.

Algorithm 4: Shake procedure

```

1 Procedure Shake( $k, x, sc, f$ );
2 Select  $\min\{k, 30\}$  vertices and move to empty group;
3 while  $k > 0$  do
4    $(u, v) \leftarrow (\text{RandVert}, \text{RandVert})$ ;
5   if  $x_u \neq x_v$  then
6     Swap( $x, u, v$ );
7      $k \leftarrow k - 1$ 
8   end
9 end
10 Computesc( $x, sc$ );
11  $f \leftarrow \text{Computeobj}(x)$ 

```

4 Computational results

Results reported in this section are obtained on an Intel x64 based computer with 3.4 GHZ CPU (with 4 cores) and 4Gb of RAM. The SGVNS heuristic is coded in C++ computer language.

13 benchmark instances The first set consists of 7 benchmark instances originally considered by Charon and Hudry [3] (rand100-100, rand300-100, rand500-100, rand300-5, zahn300, sym300-50, regnier300-50) and 6 instances generated by Brusco and Köhn [2] (rand200-100, rand400-100, rand100-5, rand200-5, rand400-5, rand500-5). For more details about these test instances see [3] and [2].

30 large new instances In order to compare heuristics on large problems, we introduce new CPP instances. They consist of sets with 1000, 1500 and 2000 vertices, each having 10 instances. The edge lengths are generated at random as integers uniformly distributed in the range $[-100, 100]$.

Preliminary testing and parameter values As explained earlier, our SGVNS has 4 parameters: $\alpha, k_{\min}, k_{\max}, k_{\text{step}}$. Values of the three neighborhood parameters are set as follows: $k_{\max} = \max\{100, n/5\}$, $k_{\min} = k_{\text{step}} = \max\{1, k_{\max}/50\}$.

To obtain the value of parameter α (used in the skewed phase of SGVNS), we run our heuristic 10 times on an instance with $n = 1000$ and with different values of α , i.e., $\alpha \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.08, 0.10, 0.15, 0.20\}$. Results are given in Table 1.

From the table we can conclude that the best value for parameter α is 0.02, and we use this value in the remaining tests.

Table 1 SGVNS results on large test instance using ten restarts and different values of α

α	f_{best}	f_{avg}	f_{worst}	Time
0.01	-875,776	-870,699.00	-861,370	989.96
0.02	-878,331	-877,307.50	-875,902	931.99
0.03	-878,153	-876,886.70	-875,075	878.77
0.04	-878,163	-877,169.60	-875,565	977.02
0.05	-877,970	-877,017.70	-874,985	993.88
0.06	-877,970	-877,208.70	-874,985	869.10
0.08	-877,881	-877,032.30	-875,561	996.54
0.10	-877,870	-877,097.00	-875,365	995.44
0.15	-877,713	-876,793.70	-875,263	981.28
0.20	-877,857	-876,935.50	-875,263	912.92

Table 2 Comparison of NS-R, NS-TS and SGVNS on small instances; SGVNS is restarted ten times with time limit $t_{\max} = n$

Name	Best	NS-R	NS-TS	SGVNS		
				f_{best}	f_{avg}	Time
rand100-5	-1407	-1407	-1407	-1407	-1407	0.33
rand100-100	-24,296	-24,296	-24,296	-24,296	-24,296	1.42
rand200-5	-4079	-4079	-4079	-4079	-4079	26.59
rand200-100	-74,924	-74,924	-74,924	-74,924	-74,924	12.56
rand300-5	-7732	-7723	-7729	-7732	-7728	87.12
rand300-100	-152,709	-152,709	-152,709	-152,709	-152,709	24.81
sym300-50	-17,592	-17,592	-17,592	-17,592	-17,592	143.45
regnier300-50	-32,164	-32,164	-32,164	-32,164	-32,164	3.24
zahn300	-2504	-2503	-2504	-2504	-2504	29.4
rand400-5	-12,133	-12,096	-12,120	-12,133	-12,123	206
rand400-100	-222,757	-222,647	-222,374	-222,757	-222,735	212.65
rand500-5	-17,127	-17,008	-17,086	-17,127	-17,095.5	255.29
rand500-100	-309,125	-308,620	-308,341	-309,107	-308,754	291.00

A limit on CPU time set at $t_{\max} = n$ (in seconds) is used as the stopping condition in all runs.

A requested by an anonymous referee, we also compared our SGVNS heuristic with an exact method. To this end we selected a CPLEX-LP solver for the linear formulation of the CPP, and a CPLEX quadratic solver for the equivalent quadratic program [see (CPP_e) above]. The quadratic solver required excessive computer time, and thus, was limited to very small test instances ($n \leq 15$). The SGVNS heuristic obtained the optimal solution in all cases tested ($n = 10, 15, 20, 25, 30, 35$). For example, for the test instance with $n = 35$ nodes, the CPLEX-LP solver took 4127 s to find the optimal solution, whereas SGVNS obtained the same solution in 0.03 s!

Table 3 Results on large instances, based on ten restarts for all methods and with the time limit set to number of vertices ($t_{\max} = n$)

Name	NS-R		NS-TS		SGVNS		%Im over	
	f_{best}	f_{avg}	Time	f_{best}	f_{avg}	Time	NS-R	NS-TS
r1000_01	-867,120	-864,600.33	1054.32	-866,105	-865,510.67	1033.85	-877,307.50	874.28
r1000_02	-867,506	-866,265.33	1048.33	-872,131	-867,497.67	1113.92	-880,551.90	931.99
r1000_03	-869,632	-866,454.33	1047.74	-867,505	-862,686.33	1086.74	-878,174.10	1037.07
r1000_04	-863,587	-860,479.33	1059.32	-864,306	-860,894.67	1080.27	-873,902.30	1067.97
r1000_05	-871,291	-869,459.33	1055.17	-873,648	-872,259.67	1092.70	-881,738.80	987.96
r1000_06	-883,498	-880,247.60	1043.91	-888,777	-883,757.30	1070.99	-886,735.00	916.05
r1000_07	-874,515	-873,598.33	1069.01	-874,095	-870,816.00	1056.59	-887,235.50	1131.97
r1000_08	-877,350	-873,365.00	1043.38	-878,923	-874,766.67	1081.24	-878,897.60	802.65
r1000_09	-884,951	-883,422.33	1060.23	-888,638	-886,901.00	1023.65	-891,992.40	697.56
r1000_10	-868,132	-865,049.00	1050.94	-866,735	-864,425.67	1088.51	-876,374.20	1082.99
Avg.	-872,758.2	-870,294.09	1053.24	-874,086.3	-870,951.56	1072.85	-881,230.36	953.05
r1500_01	-1,600,474	-1,594,384.33	1596.51	-1,592,399	-1,591,123.67	1782.53	-1,610,805.22	1185.06
r1500_02	-1,583,069	-1,577,906.33	1541.10	-1,584,404	-1,580,197.33	1649.66	-1,614,664.20	1231.76
r1500_03	-1,583,597	-1,581,930.00	1563.01	-1,591,449	-1,583,654.00	1657.47	-1,610,329.10	1292.90
r1500_04	-1,578,713	-1,575,948.33	1541.04	-1,590,351	-1,580,704.67	1713.55	-1,609,826.30	1123.63
r1500_05	-1,585,727	-1,582,063.33	1553.26	-1,586,615	-1,584,059.33	1601.34	-1,614,855.00	1240.69
r1500_06	-1,582,878	-1,578,623.33	1541.54	-1,588,589	-1,580,136.33	1670.01	-1,610,638.50	1326.36
r1500_07	-1,609,452	-1,601,790.00	1536.98	-1,617,723	-1,607,697.00	1656.17	-1,636,085.00	1253.96
r1500_08	-1,565,381	-1,562,638.00	1545.45	-1,571,224	-1,563,231.67	1591.34	-1,586,554.00	1286.64
r1500_09	-1,600,449	-1,598,608.33	1557.15	-1,607,941	-1,600,318.33	1642.45	-1,612,040.60	1251.96
r1500_10	-1,593,909	-1,588,740.33	1552.03	-1,585,602	-1,582,132.33	1644.01	-1,595,823.20	1042.75
Avg.	-1,588,364.9	-1,584,263.23	1552.81	-1,591,629.7	-1,585,325.47	1660.85	-1,608,854.25	1223.57

Table 3 continued

Name	NS-R			NS-TS			SGVNS			%Im over	
	f_{best}	f_{avg}	Time	f_{best}	f_{avg}	Time	f_{best}	f_{avg}	Time	NS-R	NS-TS
r2000_01	-2,467,372	-2,460,571.60	2081.64	-2,476,734	-2,459,899.60	2237.85	-2,493,143.00	-2,485,535.50	1596.02	1.03	0.66
r2000_02	-2,463,804	-2,451,838.00	2062.19	-2,467,493	-2,447,006.10	2176.77	-2,484,001.00	-2,473,849.60	1451.65	0.81	0.66
r2000_03	-2,462,756	-2,458,149.90	2062.58	-2,477,969	-2,460,834.60	2152.34	-2,490,379.00	-2,481,812.60	1569.89	1.11	0.50
r2000_04	-2,469,300	-2,459,721.00	2051.13	-2,478,567	-2,461,141.70	2129.76	-2,497,504.00	-2,485,469.90	1633.03	1.13	0.76
r2000_05	-2,445,974	-2,435,498.50	2107.07	-2,445,534	-2,436,632.40	2365.45	-2,472,230.00	-2,459,713.90	1641.98	1.06	1.08
r2000_06	-2,451,637	-2,436,883.30	2085.50	-2,449,319	-2,436,018.60	2132.28	-2,473,995.00	-2,462,179.60	1684.78	0.90	1.00
r2000_07	-2,459,744	-2,449,809.70	2072.86	-2,456,281	-2,448,892.00	2290.89	-2,483,346.00	-2,472,279.20	1760.67	0.95	1.09
r2000_08	-2,440,670	-2,431,941.10	2065.87	-2,439,846	-2,423,618.60	1115.77	-2,467,568.00	-2,451,094.40	1818.02	1.09	1.12
r2000_09	-2,468,137	-2,458,719.80	1062.17	-2,461,927	-2,454,540.10	1124.97	-2,495,666.00	-2,479,213.90	1740.09	1.10	1.35
r2000_10	-2,415,438	-2,409,663.80	1059.33	-2,422,183	-2,411,409.50	1173.93	-2,453,464.00	-2,437,303.70	1663.52	1.55	1.27
Avg.	-2,454,483.2	-2,445,279.67	1871.03	-2,457,585.3	-2,443,999.32	1890.00	-2,481,129.60	-2,468,845.23	1655.96	1.07	0.95

Comparison with state-of-the-art heuristics In this subsection we compare results obtained by our SGVNS with those from the literature. As alluded to earlier, the state-of-the-art comprises the “neighborhood search” heuristics referred to as NS-R and NS-TS in [2]. In Table 2, we compare SGVNS with these two heuristics on the 13 small test instances. The second column reports the best known values obtained by our SGVNS. Objective function values obtained by NS-R and NS-TS are given in columns 3 and 4, respectively. The next 3 columns give results obtained by SGVNS: the best objective value in 10 restarts, the average value and the average CPU time spent before the best claimed solutions were found. Note that the running times of the other two heuristics were set to 500 s in [2].

It appears that results of equivalent quality are obtained by all three heuristics for these smaller instances, although SGVNS is slightly better on some. However, we shall see that for the larger instances, SGVNS reports significantly better results.

In Table 3 the same methods are compared on our new large size test instances. This time the best in 10, average in 10 and average cpu times for all three heuristics are reported. In the last two columns we give the % improvement of SGVNS over the other two state-of-the-art heuristics.

It appears that in all 30 large instances, SGVNS solutions were of better quality and obtained in less cpu time. Therefore, our new heuristic may be considered as the state-of-the-art for solving the CPP.

5 Conclusions

In this paper we develop a Skewed general variable neighborhood search (SGVNS) heuristic for solving the clique partitioning problem (CPP). This new approach combines the diversification provided by “skewed” variable neighborhood search with the intensification of variable neighborhood descent (VND), the local search step in “general” variable neighborhood search.

The combined SGVNS approach was recently used with success on the related maximally diverse grouping problem (Brimberg et al. [1]). We obtain similar success here on the CPP, with significant improvement on solution quality reported on several large scale instances compared to the state of the art.

Future research directions include the design and testing of different neighborhoods within VND, and the application of SGVNS to other combinatorial problems.

Acknowledgements This research has been supported in part by a Natural Sciences and Engineering Research Council of Canada Discovery Grant (NSERC #205041-2008) and by Serbian Ministry of Sciences, Project #174010. This work is also partly supported by RSF (Russian Federation) grant 14-41-00039.

References

1. Brimberg, J., Mladenović, N., Urošević, D.: Solving the maximally diverse grouping problem by skewed general variable neighborhood search. *Inform Sci* **295**, 650–675 (2015)
2. Brusco, M.J., Köhn, H.-F.: Clustering qualitative data based on binary equivalence relations: neighborhood search heuristics for the clique partitioning problem. *Psychometrika* **74**(4), 685–703 (2009)

3. Charon, I., Hudry, O.: Noising methods for a clique partitioning problem. *Discrete Appl. Math.* **154**(5), 754–769 (2006)
4. De Amorim, S.G., Barthélemy, J.P., Ribeiro, C.C.: Clustering and clique partitioning: simulated annealing and tabu search approaches. *J. Classif.* **9**, 17–41 (1992)
5. Desrosiers, J., Mladenović, N., Villeneuve, D.: Design of balanced MBA student teams. *J. Oper. Res. Soc.* **56**, 60–66 (2005)
6. Hansen, P., Mladenović, N.: Variable neighbourhood search. *Ann. Oper. Res.* **175**, 367–407 (2010)
7. Hettich, S., Pazzani, M.J.: Min lessons learned at the national science foundation. In: *Proceedings of the KDD'06*, pp. 862–871. ACM, New York (2006)
8. Kral, J.: To the problem of segmentation of a program. *Inf. Process. Mach.* **2**, 116–127 (1965)
9. Lotfi, V., Cervený, R.: A final exam scheduling package. *J. Oper. Res. Soc.* **42**, 205–216 (1991)
10. Marcotorchino, F., Michaud, P.: Heuristic approach to the similarity aggregation problem. *Methods Oper. Res.* **43**, 395–404 (1981)
11. Mladenović, N., Hansen, P.: Variable neighborhood search. *Comput. Oper. Res.* **24**, 1097–1100 (1997)
12. Oosten, M., Rutten, J.H.G.C., Spijksma, F.C.R.: The clique partitioning problem: facets and patching facets. *Networks* **38**(4), 209–226 (2001)
13. Régnier, S.: Sur quelques aspects mathématiques des problèmes de classification automatique. *I.C.C. Bulletin* **4**, 175–191 (1965)
14. Urošević, D.: Variable neighborhood search for maximum diverse grouping problem. *Yugosl. J. Oper. Res.* **24**(1), 21–33 (2014)
15. Wang, H., Alidaee, B., Glover, F., Kochenberger, G.: Solving group technology problems via clique partitioning. *Int. J. Flex. Manuf. Syst.* **18**, 77–97 (2006)
16. Wang, H., Obremski, T., Alidaee, B., Kochenberger, G.: Clique partitioning for clustering: a comparison with k -means and latent class analysis. *Commun. Stat. Simul. Comput.* **37**, 1–13 (2008)
17. Weitz, R.R., Lakshminarayanan, S.: An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design, and exam scheduling. *Omega* **25**(4), 473–482 (1997)
18. Weitz, R.R., Lakshminarayanan, S.: An empirical comparison of heuristic methods for creating maximally diverse groups. *J. Oper. Res. Soc.* **49**(6), 635–646 (1998)