Neighborhood Combination for Unconstrained Binary Quadratic Problems

Zhipeng Lü[1], Fred Glover[2], and Jin-Kao Hao[1]
lu@info.univ-angers.fr, glover@opttek.com, hao@info.univ-angers.fr

[1] LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers Cedex 01, France
[2] OptTek Systems, Inc., 2241 17th Street Boulder, CO 80302, USA

Abstract. We present an experimental analysis of neighborhood combinations for local search based metaheuristic algorithms, using the Unconstrained Binary Quadratic Programming (UBQP) problem as a case study. The goal of the analysis is to help understand why, when and how some neighborhoods can be favorably combined to increase their search power. Our study investigates combined neighborhoods with two types of moves for the UBQP problem within a Tabu Search algorithm to determine which strategies for combining neighborhoods prove most valuable.
keywords: UBQP, Tabu Search, Neighborhood Combination, Neighborhood Analysis.

1 Introduction

Neighborhood search or local search is known to be a highly effective metaheuristic framework for solving a large number of constraint satisfaction and optimization problems. By defining a neighborhood and starting from an initial solution, local search progressively explores the neighborhood of the present solution for improvement. In this way, the current solution is iteratively replaced by one of its neighbors (often improving) until a specific stop criterion is satisfied.

One of the most important features of local search is the definition of its neighborhood. In general, good neighborhoods offer a high search capability and consequently lead to good results largely independent of the initial solution while the search performance induced by weak neighborhoods is often highly correlated to the initial solution [37]. Generally, a local optimum for one neighborhood is not necessarily a local optimum for another. Therefore, it is possible and interesting to create more powerful combined neighborhoods.

Using the Unconstrained Binary Quadratic Programming (UBQP) problem as a case study, we present in this work several combinations of neighborhoods, using one-flip and two-flip moves. The two-flip move proposed in this paper is new for the UBQP problem. To evaluate their performance, we carried out extensive experiments with a Tabu Search algorithm run on a large set of benchmark instances. Computational results show that certain combinations are superior to others.

The remaining part of this paper is organized as follows. Section 2 gives the description of the UBQP problem together with its recent advances. In Section 3, the one-flip and two-flip moves and their fast evaluation techniques are fully described. Sections 4 is dedicated to several neighborhood combinations and our Tabu Search algorithm. In Section 5, we present our computational comparison on these neighborhoods and their combinations, and draw inferences from these findings about the factors that cause certain neighborhood combinations to be effective or ineffective. Finally in Section 6, we provide some conclusions and discuss some important issues related to this work.

2 Unconstrained Binary Quadratic Programming

The unconstrained binary quadratic programming problem may be written as:

$$\begin{array}{l}\text{UBQP: Maximize } x_o = xQx' \\ x \text{ binary}\end{array}$$

where Q is an $n \times n$ matrix of constants and x is an n-vector of binary (zero-one) variables.

In recent decades, the UBQP formulation has attracted wide attention for its ability to represent a wide range of important problems, including those from social psychology [20], financial analysis [27, 31], computer aided design [26], traffic management [11, 41], machine scheduling [1], cellular radio channel allocation [9] and molecular conformation [40]. Moreover, the application potential of UBQP is much greater than might be imagined, due to the possibilities of imposing quadratic infeasibility constraints into the objective function in an explicit manner. For instance, many combinatorial optimization problems pertaining to graphs such as determining maximum cliques, maximum cuts, maximum vertex packing, minimum coverings, maximum independent sets, maximum independent weighted sets are known to be capable of being formulated by the UBQP problem as documented in papers of [38, 39]. A review of additional applications and formulations can be found in [2, 24, 25, 28].

For the UBQP problem, many exact algorithms have been proposed. The most successful approaches include those of [5, 21, 38]. However, due to its computational complexity, exact algorithms can only solve instances of small size (with 100 variables). Therefore, a large number of heuristic and metaheuristic solution procedures have been reported in the literature to handle large instances. Some representative examples include local search based approaches such as Simulated Annealing [3, 7, 22] and Tabu Search [7, 17, 18, 35, 36], population-based approaches such as Evolutionary Algorithms [8, 23, 29, 32], Scatter Search [4] and Memetic Algorithms [33].

3 Neighborhood Moves and Fast Evaluation

In a local search procedure, applying a move mv to a candidate solution x leads to a new solution denoted by $x \bigoplus mv$. Let $M(x)$ be the set of all possible moves which can be applied to x, then the neighborhood NB of x is defined by: $NB(x) = \{x \bigoplus mv | mv \in M(x)\}$. For the UBQP problem, we use two distinct *moves* denoted by one-flip and two-flip moves. In the following, we respectively denote the neighborhoods with one-flip and two-flip moves N_1 and N_2.

3.1 One-flip move

The one-flip move defining neighborhood N_1 complements (flips) a chosen binary variable x_i by subtracting its current value from 1, i.e., the value of variable x_i becomes $1 - x_i$ after a one-flip move. One-flip is widely used in local search algorithms for binary problems such as UBQP, multi-dimensional knapsack and satisfiability problems.

Let $N = \{1, \ldots, n\}$ denote the index set for components of the x vector. We preprocess the matrix Q to put it in lower triangular form by redefining (if necessary) $q_{ij} = q_{ij} + q_{ji}$ for $i > j$, which is implicitly accompanied by setting $q_{ji} = 0$ (though these 0 entries above the main diagonal are not stored or accessed). Let Δ_i be the move value of flipping the variable x_i, and let $q_{(i,j)}$ be a shorthand for denoting q_{ij} if $i > j$ and q_{ji} if $j > i$. Then each move value can be calculated in linear time using the formula:

$$\Delta_i = (1 - 2x_i)(q_{ii} + \sum_{j \in N, j \neq i, x_j = 1} q_{(i,j)}) \tag{1}$$

For large problem instances, it is imperative to be able to rapidly determine the effect of a move on the objective function x_o. For this purpose, we employ a fast incremental evaluation technique first introduced by [17] and enhanced by [13] to exploit an improved representation and to take advantage of sparse data - a characteristic of many real world problems. The procedure maintains a data structure that stores the move value (change in x_o) for each possible move, and employs a streamlined calculation for updating this data structure after each iteration.

Moreover, it is not necessary to recalculate all the move values after a move. Instead, one needs just to update a subset of move values affected by the move. More precisely, it is possible to update the move values upon flipping a variable x_i by performing the following abbreviated calculation:

1. $\Delta_i = -\Delta_i$
2. For each $j \in N - \{i\}$,
   $\Delta_j = \Delta_j + \sigma_{ij}\ q_{(i,j)}$
   where $\sigma_{ij} = 1$ if $x_j = x_i$, $\sigma_{ij} = -1$ otherwise.

We employ the convention that x_i represents x_i's value before being flipped.

3.2 Two-flip move

In the case of a two-flip neighborhood N_2, we are interested in the change in x_o that results by flipping 2 variables, x_i and x_j, and will refer to this change by δ_{ij}. It is convenient to think of the two-flip process as a combination of two single one-flip moves, and we can derive δ_{ij} using the one-flip move values Δ_i and Δ_j as follows (supposing $i > j$):

$$\delta_{ij} = \Delta_i + \Delta_j + \lambda_{ij}\ q_{(i,j)} \tag{2}$$

where $\lambda_{ij} = 1$ if $x_i = x_j$ and $\lambda_{ij} = -1$ otherwise.

It is easy to observe that the size of neighborhood N_2 is bounded by $O(n^2)$. After a two-flip move is performed (suppose variables x_i and x_j are flipped), we need only update the one-flip delta array Δ that is affected by this move. Specifically, the following abbreviated calculation can be performed:

1. $\Delta_i = -(\Delta_i + \sigma_{ij}\ q_{(i,j)})$
2. $\Delta_j = -(\Delta_j + \sigma_{ij}\ q_{(i,j)})$
3. For each $k \in N - \{i, j\}$,
   $\Delta_k = \Delta_k + \sigma_{ik}\ q_{(i,k)} + \sigma_{jk}\ q_{(j,k)}$
   where $\sigma_{uv} = 1$ if $x_u = x_v$ $(u, v = \{i, j, k\})$, $\sigma_{uv} = -1$ otherwise.

Here x_i and x_j represent x_i and x_j's values before being flipped.

One finds that the complexity of this updating rule is $O(n)$, i.e., at most n delta values are recalculated each time. Accompanying this updating rule, it is possible to introduce additional data structures to speed up the process of identifying the best two-flip move for the next iteration. Interested readers are referred to [14] for more details.

In spite of the linear time complexity of the updating rule after a move is performed, it is still too time-consuming to examine all the two-flip moves using formula (2) since the two-flip neighborhood N_2 has $n(n-1)/2$ neighbors at each iteration. To overcome this obstacle, we employ a *candidate list strategy* to reduce the number of candidates in the neighborhood by examining only a small subset of all the possible two-flip moves. Specifically, at each iteration, we sort all the one-flip Δ values in a decreasing order. Then, the two-flip move that flips x_i and x_j will be considered only if the values of both Δ_i and Δ_j ranks the first β best. In this paper, we empirically set $\beta = 3\sqrt{n}$ which gives satisfying results without sacrificing solution quality. The greater the value of β, the greater will be the number of two-flip neighborhood moves examined and the amount of CPU time required. Notice that if this candidate list strategy is disabled in N_2, the computation will be greater than that required by the one-flip neighborhood N_1.

4 Neighborhood Combinations and Algorithm

4.1 Neighborhood Combinations

In order to increase the search capability of single neighborhoods, it has become a popular practice to combine two or more different neighborhoods. The advantage of such an approach was demonstrated using a tabu search strategic oscillation design in [16], and additional variants of strategic oscillation for transitioning among alternative neighborhoods are discussed in [12]. More recently, the metaheuristic approach called Variable Neighborhood Search in [34] has effectively used a transition scheme that always returns to the simplest neighborhood when improvement occurs, while the transition scheme that cycles through higher levels before returning to the simplest (also studied in [16]) was examined in [10] and elaborated more fully in the metaheuristic context in [19].

Several ways exist for combining different neighborhoods. In this paper, we focus on two of them: neighborhood union and token-ring search [10, 30].

We define two forms of neighborhood union: *strong neighborhood union* and *selective neighborhood union*. For strong neighborhood union, denoted by $N_1 \sqcup N_2$, the algorithm picks each move (according to the algorithm's selection criteria) from all the N_1 and N_2 moves. For selective neighborhood union, denoted by $N_1 \cup N_2$, the search algorithm selects one of the two neighborhoods to be used at each iteration, choosing the neighborhood N_1 with a predefined probability p and choosing N_2 with probability 1-p. An algorithm using only N_1 or N_2 is of course a special case of an algorithm using $N_1 \cup N_2$ where p is set to be 1 and 0 respectively.

In token-ring search, the neighborhoods are alternated, applying the currently selected neighborhood without interruption, starting from the local optimum of the previous neighborhood, until no improvement is possible. More precisely, the search procedure uses one neighborhood until a *best* local optimum is determined, subject to time or iteration limits imposed on the search (For metaheuristic searches, this may not be the first local optimum encountered.) The *best* local optimum here denotes the best solution found so far by the current search. Then the method switches to the other neighborhood, starting from this local optimum, and continues the search in the same fashion. The search comes back to the first neighborhood at the end of the second neighborhood exploration, repeating this process until no improvement is possible. The token-ring search of two neighborhoods can be denoted as $N_1 \rightarrow N_2$ (starting from N_1) or $N_2 \rightarrow N_1$ (starting from N_2). More details are given in [30].

4.2 Tabu Search Algorithm

For the purpose of studying the different neighborhoods and their combinations, we implement a simple Tabu Search (TS) algorithm [15]. TS typically incorporates a tabu list as a "recency-based" memory structure to assure that solutions visited within a certain span of iterations, called the tabu tenure, will not be revisited. The approach is designed to introduce vigor into the search by also forbidding moves leading to related solutions that share certain attributes (values of variables) in common with the visited solutions. In present implementation, each time a variable x_i is flipped, this variable enters into the tabu list (an n-vector $TabuTenure$) and cannot be flipped for the next $TabuTenure(i)$ iterations ($TabuTenure(i)$ is the "tabu tenure"). For the current study, we elected to set

$$TabuTenure(i) = C + rand(10) \tag{3}$$

where C is a given constant and rand(10) takes a random value from 1 to 10.

For the one-flip neighborhood, our TS algorithm then restricts consideration to variables not forbidden by the tabu list, and selects a variable to flip that produces the largest Δ_i value (thus improving x_o if this value is positive). In the case that two or more moves have the same best move value, a random best move is selected. For the two-flip neighborhood, a move is declared tabu if and only if both two flipping variables are in tabu status.

However, some of those neighborhood solutions forbidden by the tabu list might be of excellent quality and might not have been visited. To mitigate this problem, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution.

In the case that TS procedure is applied to a token ring search (denoted $N_1 \rightarrow N_2$ for our two neighborhoods case), we start the TS procedure with neighborhood N_1. Since we need to search the two neighborhoods alternately, the application of TS to a single neighborhood stops when the best solution cannot be improved within a given number θ of moves and we call this number the *improvement cutoff* of TS, which we empirically set to be a relatively small value (50,000 for all the tested instances).

5 Experimental Results

In this Section, we show computational results for our simple TS algorithm using the following neighborhoods and neighborhood combinations: N_1 (one-flip), N_2 (two-flip), $N_1 \cup N_2$ (selective union) with $p = 0.5$, $N_1 \sqcup N_2$ (strong union) and $N_1 \rightarrow N_2$ (token-ring).

5.1 Test Instances and Experimental Protocol

Two sets of test problems are considered in our experiments. The first set of benchmarks is composed of the 10 largest instances of size $n = 2500$ introduced in [7] and available in the ORLIB [6]. These instances are used in the literature by many authors (e.g., [7, 22, 33, 35, 36]). The second set of benchmarks consists of a set of 15 randomly generated large problem instances named p3000.1,...,p5000.5 with sizes ranging from n=3000 to 5000 [35, 36]. These instances are available at: http://www.soften.ktu.lt/∼gintaras/ubqop_its.html.

Our algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.66GHz CPU and 512M RAM. For each run of the TS algorithm, the initial solution is generated randomly, i.e., each variable x_i receives a random value of 0 or 1 with equal chance. Given this stochastic nature of our TS procedure, each problem instance is independently solved 20 times. To make the comparison as fair as possible, all the experiments use the same CPU time limits: for the 10 Beasley instances with 2500 variables, the CPU time limit is set to be 1000 seconds while it is set to be 2000 seconds for other 15 larger instances.

5.2 Computational Comparison

Table 1. Results of the TS algorithm on the 10 Beasley instances with size n=2,500 within 1000 seconds.

instance	dens	f_{prev}	solution gaps to f_{prev} ($f_{prev} - \overline{f}$)				
			N_1	N_2	$N_1 \cup N_2$	$N_1 \sqcup N_2$	$N_1 \rightarrow N_2$
b2500.1	0.1	1515944	0	4.2	0	94.0	0
b2500.2	0.1	1471392	0	12.1	0	65.1	0
b2500.3	0.1	1414192	94.1	1.4	0	301.2	0
b2500.4	0.1	1507701	0	0	0	0	0
b2500.5	0.1	1491816	0	0	0	0	0
b2500.6	0.1	1469162	0	1.3	0	82.1	0
b2500.7	0.1	1479040	35.7	1.3	0	122.5	0
b2500.8	0.1	1484199	0	8.2	0	10.2	3.5
b2500.9	0.1	1482413	0	10.9	0	1.9	0
b2500.10	0.1	1483355	0	4.0	0	0	0
average			12.98	4.34	0	67.7	0.35

Average Results Comparison Table 1 shows the computational statistics of the TS algorithm on the 10 Beasley instances with 2500 variables. Columns 2 and 3 respectively give the density (dens) and the best known objective values (f_{prev}) obtained from the literature. Columns 4 to 8 give the solution gap to the best solutions for each neighborhood and neighborhood combination. For each instance, the solution gap in Table 1 is represented as $f_{prev} - \overline{f}$, where $\overline{f}$ is the average objective value obtained by 20 independents runs and f_{prev} represents the previous best known objective value. The overall results, averaged over 10 instances, are presented in the last row.

From Table 1, we observe that neighborhood N_1 reaches the previous best known results very stably for 8 of the 10 instances while it performs quite poorly on other two cases. On the other hand, neighborhood N_2 can obtain the previous best known results each time only for two instances, but obtains optimal or near-optimal solutions with quite small variance for other cases. In terms of the average gaps to the previous best solutions, neighborhood N_2 slightly outperforms N_1 for these 10 test problems.

When comparing the three neighborhood combinations $N_1 \cup N_2$ (with $p = 0.5$), $N_1 \sqcup N_2$ and $N_1 \rightarrow N_2$ with each other, one finds that the selective union $N_1 \cup N_2$ and the token-ring search $N_1 \rightarrow N_2$ are superior to the strong union $N_1 \sqcup N_2$, as well as the single neighborhoods N_1 and N_2. One also observes that the strong union $N_1 \sqcup N_2$ performs much worse than the single neighborhood N_1, implying that the strong union is not an appropriate way of combination for these two neighborhoods. For each pairwise comparison of these neighborhoods, we performed a 95% confidence t-test to compare their solution quality, leading to the following ranking of the neighborhoods: for single neighborhoods $N_2 > N_1$ while $N_1 \cup N_2 > N_1 \rightarrow N_2 > N_1 \sqcup N_2$ for neighborhood combinations.

Table 2. Results of the TS algorithm on the 15 large random instances with variables ranging from 3000 to 5000 within 2000 seconds.

instance	dens	f_{prev}	solution gaps to f_{prev} ($f_{prev} - \overline{f}$)				
			N_1	N_2	$N_1 \cup N_2$	$N_1 \sqcup N_2$	$N_1 \rightarrow N_2$
p3000.1	0.5	3931583	319.8	8.8	103.4	1866.7	435.0
p3000.2	0.8	5193073	418.6	103.5	193.2	214.7	120.7
p3000.3	0.8	5111533	482.7	637.6	488.7	508.9	607.2
p3000.4	1.0	5761822	77.0	38.6	0	630.1	57.7
p3000.5	1.0	5675625	460.4	385.3	223.8	738.0	655.0
p4000.1	0.5	6181830	0	15.2	0	1249.3	0
p4000.2	0.8	7801355	1732.1	1364.5	402.2	2683.4	1622.7
p4000.3	0.8	7741685	1427.9	474.1	445.1	1742.0	936.1
p4000.4	1.0	8711822	1516.3	276.8	438.1	1954.1	1359.4
p4000.5	1.0	8908979	2979.9	372.5	397.4	3053.2	2723.7
p5000.1	0.5	8559355	2957.9	1287.3	1153.6	4118.4	2365.4
p5000.2	0.8	10836019	3561.5	2232.6	2716.5	3839.8	3263.0
p5000.3	0.8	10489137	8451.0	4156.4	3054.0	9634.5	3124.5
p5000.4	1.0	12252318	4760.2	3261.0	2215.4	9276.1	3416.3
p5000.5	1.0	12731803	6327.0	1369.3	1472.8	4863.2	6093.6
average			2364.82	1065.57	886.95	3091.49	1785.35

Similarly, the computational results of the TS algorithm on the 15 larger and denser random instances are shown in Table 2. The symbols are the same as those in Table 1. Once again, we observe that neighborhood N_2 outperforms N_1 except for two instances (p3000.4 and p4000.1) in terms of the average gaps to the previous best known objective values. In addition, the selective union $N_1 \cup N_2$ is superior to other two neighborhood combinations. We also performed a 95% confidence t-test to compare different neighborhoods and observed that $N_2 > N_1$ for single neighborhoods while $N_1 \cup N_2 > N_1 \rightarrow N_2 > N_1 \sqcup N_2$ for neighborhood combinations. These results coincide well with the results observed on the 10 Beasley instances with 2500 variables.

Best Results Comparison We now turn our attention to the *best results* that the TS algorithm obtains in the preceding experiments. Note that there is no difficulty to obtain the previous best known results for each neighborhood or neighborhood combination for all the 10 instances with 2500 variables. Thus, we only list in Table 3 the best results of the TS algorithm on the 15 larger instances. Columns 2 and 3 recall the density (dens) and the best known objective values (f_{prev}) obtained from the literature. Columns 4 to 8 give the solution gap to the best solutions for each neighborhood and neighborhood combination, where f_{best} represents the best objective value obtained over 20 independents runs.

Table 3. Best results of the TS algorithm over 20 independent runs.

instance	dens	f_{prev}	solution gaps to f_{prev} $(f_{prev} - f_{best})$				
			N_1	N_2	$N_1 \cup N_2$	$N_1 \sqcup N_2$	$N_1 \rightarrow N_2$
p3000.1	0.5	3931583	0	0	0	0	0
p3000.2	0.8	5193073	0	0	0	0	0
p3000.3	0.8	5111533	0	0	0	0	0
p3000.4	1.0	5761822	0	0	0	0	0
p3000.5	1.0	5675625	0	0	0	0	0
p4000.1	0.5	6181830	0	0	0	0	0
p4000.2	0.8	7801355	0	0	0	1686	0
p4000.3	0.8	7741685	0	0	0	0	0
p4000.4	1.0	8711822	0	0	0	0	0
p4000.5	1.0	8908979	0	0	0	0	0
p5000.1	0.5	8559355	0	-325	-325	0	0
p5000.2	0.8	10836019	582	65	0	582	0
p5000.3	0.8	10489137	354	148	148	683	663
p5000.4	1.0	12252318	608	0	0	2400	0
p5000.5	1.0	12731803	1025	0	0	0	0
average			256.9	-11.2	-17.7	366.5	66.3

Once again, the overall results averaged over 15 instances are presented in the last row.

Table 3 shows that the selective union $N_1 \cup N_2$ performs much better than the strong union $N_1 \sqcup N_2$, the token-ring search $N_1 \rightarrow N_2$ and the single neighborhoods N_1, and even slightly better than N_2. One also observes that for the 15 larger instances, the TS algorithm with $N_1 \cup N_2$ matches the previous best results for 13 of them, while getting a worse result only for one instance and a better result for the remaining one. It should be noticed that the selective union $N_1 \cup N_2$ and the single neighborhood N_2 both improve the best result obtained by [36] for instance p5000.1, showing the advantage of the newly introduced neighborhood N_2 over N_1 and the combination mechanism of selective union. According to these results, we have the following ranking of the neighborhoods: $N_1 \cup N_2 > N_2 > N_1 \rightarrow N_2 > N_1 > N_1 \sqcup N_2$. The trends of the best costs perfectly match those of the average costs mentioned above for the considered instances.

Results Analysis The preceding computational results show that for the three neighborhood combinations of N_1 and N_2, the selective union $N_1 \cup N_2$ produces much better results than other combinations. These results prompt us to focus on investigating the best and worst neighborhood combinations: $N_1 \cup N_2$ and $N_1 \sqcup N_2$. In this section, we attempt to explain what causes the effectiveness and weakness of these two neighborhood unions and show evidence for this phenomenon in terms of three evaluation criteria. For this purpose, we employ a steepest descent (SD) algorithm for this experiment, where we disable the tabu list of our TS algorithm and the current solution is repeatedly replaced by a *best improving* solution in its neighborhood until no improving neighbor exists. The experiment is carried out on the large instance p5000.3 (very similar results are observed for other instances).

In [30], three evaluation criteria were employed to characterize the search capacity of a neighborhood: *percentage of improving neighbors*, *improvement strength* and *search steps*. The authors argue that good neighborhoods should have one or more of these features: high percentage of improving neighbors (for more improvement possibilities), strong improvement strength (for important improvements) and long search steps (for long term improvements).

For a candidate solution x, a given neighborhood function $NB : \mathcal{X} \to 2^{\mathcal{X}}$ and a neighborhood solution $x' \in NB(x)$, define $\Delta f = f(x') - f(x)$. These criteria are then defined as follows.

- *Improving neighbors* $I(x)$: the set of the improving neighbors in the neighborhood $NB(x)$ given by $I(x) = \{x' \in NB(x) | \Delta f > 0\}$. Therefore, the *percentage of improving neighbors* is defined as $|I(x)|/|NB(x)| \times 100\%$.
- *Improvement strength* Δf^*: the cost variation between the current solution x and a best improving neighbor given by $\Delta f^* = max\{|\Delta f| : \Delta f \in I(x)\}$.
- *Search steps*: the term *search steps* is defined as the number of iterations that the SD algorithm can run to reach a local optimum.

To calculate the values of each criterion, we run the SD algorithm for 50 independent runs respectively with $N_1 \cup N_2$ and $N_1 \sqcup N_2$. For each run, data corresponding to the above three evaluation criteria are calculated; *percentage of improving neighbors* and *improvement strength* values are collected at each iteration while *search steps* is simply the iteration number when SD stops. All the reported results correspond to the averages obtained for these 50 independent runs.

Figure 1 presents the percentage of improving neighbors for $N_1 \cup N_2$ and $N_1 \sqcup N_2$, evolving with the local search iterations. It shows that at the beginning of the local search, the percentage of improving neighbors for the strong union $N_1 \sqcup N_2$ is greater than that of the selective union $N_1 \cup N_2$. However, this trend only lasts for the first 1000 local search iterations and then the percentage of improving neighbors for $N_1 \sqcup N_2$ decreases dramatically during the following 300 iterations. On the other hand, the percentage of improving neighbors for $N_1 \cup N_2$ decreases quite slowly during the first 1500 iterations. In other words, $N_1 \cup N_2$ offers more opportunities to find improving neighbors, especially after the first iterations of the search (first 1000 iterations for this particular instance). When starting from a random initial solution even poor neighborhoods can have a certain number of improving neighbors at the first iterations while only good neighborhoods offer improving neighbors when the search progresses.

On the other hand, compared with $N_1 \sqcup N_2$, there exist long tails for the percentage of improving neighbors for the selective union $N_1 \cup N_2$, meaning that it allows the descent algorithm to run a larger number of iterations. This property is another important indicator of good neighborhoods. We argue that one neighborhood with longer search steps has more potential to improve the solution quality in the long run than one with shorter search steps.

We then evaluate the two neighborhood unions using the *improvement strength* criterion. Figure 2 presents how the *improvement strength* of each neighborhood

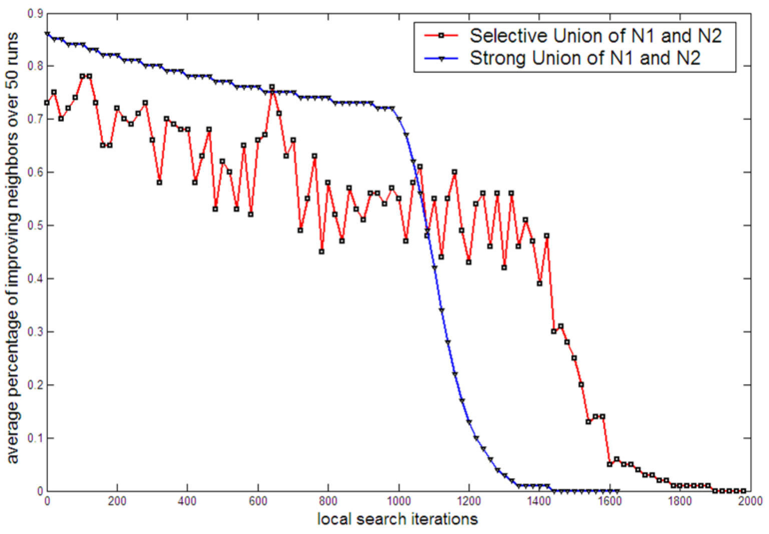

Fig. 1. The improving neighbors comparison between $N_1 \cup N_2$ and $N_1 \sqcup N_2$.

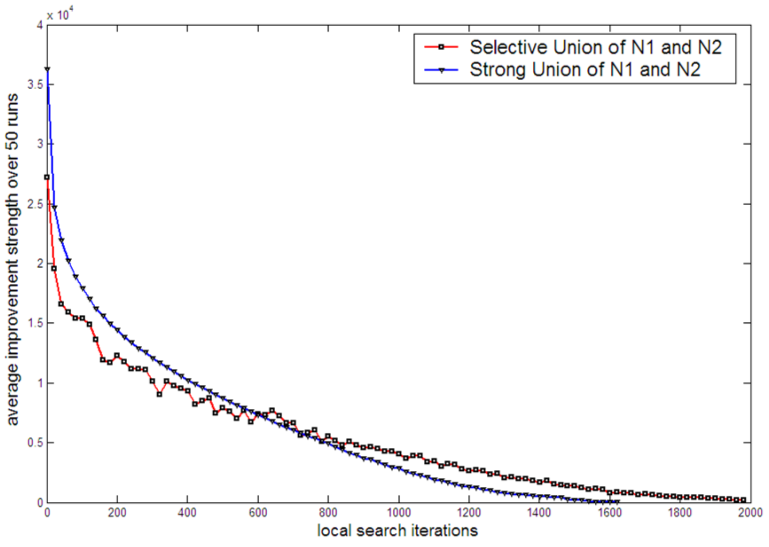

Fig. 2. The improvement strength comparison between $N_1 \cup N_2$ and $N_1 \sqcup N_2$.

evolves with the local search iterations. It shows that these two neighborhood unions have quite similar evolving trends in terms of the average improvement strength. Once again, one observes that at the beginning of the search, the improvement strength of $N_1 \sqcup N_2$ is greater than that of $N_1 \cup N_2$. However, it only lasts for a small number of iterations (the first 600 iteration in this particular case). This can be explained by the fact that $N_1 \sqcup N_2$ simultaneously considers two neighborhoods N_1 and N_2 while $N_1 \cup N_2$ only randomly chooses one neighborhood. Nevertheless, as the algorithm progresses (after the first 600 iterations), the $N_1 \cup N_2$ neighborhood offers much greater improvement strength than $N_1 \sqcup N_2$. This phenomenon correlates well with the trend of the percentage of improving neighbors.

Based on these observations, we formulate the following conclusions.

1. Neighborhood union $N_1 \cup N_2$ induces a higher percentage of improving neighbors and greater improvement strength than $N_1 \sqcup N_2$ after the first iterations of the search. As a result, $N_1 \cup N_2$ offers more choices for the search algorithm to improve the current solution at each iteration once the initial iterations are completed.
2. Neighborhood $N_1 \cup N_2$ offers improving neighbors for a larger number of iterations than $N_1 \sqcup N_2$. Consequently, local search can continue for a larger number of iterations with $N_1 \cup N_2$.
3. Although neighborhood $N_1 \sqcup N_2$ offers a higher percentage of improving neighbors and greater improvement strength during the first iterations, its improvements quickly disappear, limiting its search capability.

We also repeated this experiment with the TS algorithm described in Section 4.2 and reached similar conclusions.

6 Conclusions and Discussions

In this paper, we compare and analyze two basic neighborhoods (one-flip and two-flip) and three neighborhood combinations (*selective union*, *strong union* and *token-ring search*) for the UBQP problem. The computational results show that the best outcomes are achieved with the selective union $N_1 \cup N_2$, followed by using the new N_2 neighborhood by itself.

We employ three evaluation criteria to explain why the selective union $N_1 \cup N_2$ performs much better than the strong union $N_1 \sqcup N_2$, yielding an experimental analysis that sheds light on the relative advantages and weaknesses of the neighborhoods N_1 and N_2 and various possibilities for combining them. Our findings are anticipated to have useful implications for combining neighborhoods in other applications, particularly when presented a choice between the use of selective unions and strong unions.

Some important questions remain.

1. These results are based on *random* instances. It would be interesting to know whether these results would be confirmed for problems that exhibit special

structures of various types. To this end, a sequel to this study will carry out additional experiments using more diverse instances transformed from other problems.
2. It would be useful to identify the conditions under which a particular neighborhood or a neighborhood combination is preferable.
3. More importantly, it would be valuable to explore higher order neighborhood moves (e.g., three-flip or even higher flip moves). As observed in [14], there exits a natural way to extend the above mentioned fast two-flip move evaluation techniques to these higher order moves.
4. It would be worthwhile to investigate other ways of combining the neighborhoods, particularly with the inclusion of higher order flip moves. For example, we may consider "conditional" combinations where moves from a lower order neighborhood pass certain screening criteria as a foundation for becoming components of moves in higher order neighborhoods.

We anticipate that answers to these issues will provide information that will be valuable for the design of improved algorithms. Finally, given that the neighborhood combination strategies and the neighborhood evaluation criteria discussed in this paper is independent of the UBQP problem, they can be used to evaluate neighborhood relations of other combinatorial optimization problems.

Acknowledgement

We would like to thank the anonymous referees for their helpful comments and questions. The work is partially supported by a "Chaire d'excellence" from "Pays de la Loire" Region (France) and regional MILES (2007-2009) and RaDaPop projects (2008-2011).

References

1. Alidaee B, Kochenberger GA, Ahmadian A (1994) 0-1 quadratic programming approach for the optimal solution of two scheduling problems. International Journal of Systems Science 25:401–408
2. Alidaee B, Kochenberger GA, Lewis K, Lewis M, Wang H (2008) A new approach for modeling and solving set packing problems. European Journal of Operational Research 86(2):504–512
3. Alkhamis TM, Hasan M, Ahmed MA (1998) Simulated annealing for the unconstrained binary quadratic pseudo-boolean function. European Journal of Operational Research 108:641–652
4. Amini M, Alidaee B, Kochenberger GA (1999) A scatter search approach to unconstrained quadratic binary programs, McGraw-Hill, New York, NY, pp 317–330. New Methods in Optimization
5. Barahona F, Jünger M, Reinelt G (1989) Experiments in quadratic 01 programming. Math Program 44:127137
6. Beasley JE (1996) Obtaining test problems via internet. Journal of Global Optimization 8:429–433

7. Beasley JE (1998) Heuristic algorithms for the unconstrained binary quadratic programming problem. Working Paper, The Management School, Imperial College, London, England
8. Borgulya I (2005) An evolutionary algorithm for the binary quadratic problems. Advances in Soft Computing 2:3–16
9. Chardaire P, Sutter A (1994) A decomposition method for quadratic zero-one programming. Management Science 41(4):704–712
10. Di Gaspero L, Schaerf A (2006) Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modeling and Algorithms 5(1):65–89
11. Gallo G, Hammer P, Simeone B (1980) Quadratic knapsack problems. Mathematical Programming 12:132–149
12. Glover F (1996) Tabu Search and adaptive memory programming – advances, applications and challenges, Kluwer Academic Publishers, pp 1–75. Interfaces in Computer Science and Operations Research
13. Glover F, Hao JK (2009a) Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems. To appear in International Journal of Metaheuristics 1(1)
14. Glover F, Hao JK (2009b) Fast 2-flip move evaluations for binary unconstrained quadratic optimization problems. To appear in International Journal of Metaheuristics
15. Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers, Boston
16. Glover F, McMillan C, Glover R (1984) A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. Journal of Operations Management 4(2):113–128
17. Glover F, Kochenberger GA, Alidaee B (1998) Adaptive memory tabu search for binary quadratic programs. Management Science 44:336–345
18. Glover F, Lü Z, Hao JK (2009) Diversification-driven tabu search for unconstrained binary quadratic problems. To appear in 4OR
19. Goëffon A, Richer JM, Hao JK (2008) Progressive tree neighborhood applied to the maximum parsimony problem. IEEE/ACM Transactions on Computational Biology and Bioinformatics 5(1):136–145
20. Harary F (1953) On the notion of balanced of a signed graph. Michigan Mathematical Journal 2:143–146
21. Helmberg C, Rendl F (1998) Solving quadratic (0,1)-problem by semidefinite programs and cutting planes. Math Program 82:291–315
22. Katayama K, Narihisa H (2001) Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. European Journal of Operational Research 134:103–119
23. Katayama K, Tani M, Narihisa H (2000) Solving large binary quadratic programming problems by an effective genetic local search algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'00), Morgan Kaufmann, pp 643–650
24. Kochenberger GA, Glover F, Alidaee B, Rego C (2004) A unified modeling and solution framework for combinatorial optimization problems. OR Spectrum 26:237–250
25. Kochenberger GA, Glover F, Alidaee B, Rego C (2005) An unconstrained quadratic binary programming approach to the vertex coloring problem. Annals of Operations Research 139:229–241
26. Krarup J, Pruzan A (1978) Computer aided layout design. Mathematical Programming Study 9:75–94

27. Laughunn DJ (1970) Quadratic binary programming. Operations Research 14:454–461
28. Lewis M, Kochenberger GA, Alidaee B (2008) A new modeling and solution approach for the set-partitioning problem. Computers and Operations Research 35(3):807–813
29. Lodi A, Allemand K, Liebling TM (1999) An evolutionary heuristic for quadratic 0-1 programming. European Journal of Operational Research 119(3):662–670
30. Lü Z, Hao JK, Fred G (2009) Neighborhood analysis: a case study on curriculum-based course timetabling. Working Paper, University of Angers, France
31. McBride RD, Yormark JS (1980) An implicit enumeration algorithm for quadratic integer programming. Management Science 26:282–296
32. Merz P, Freisleben B (1999) Genetic algorithms for binary quadratic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99), Morgan Kaufmann, pp 417–424
33. Merz P, Katayama K (2004) Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78:99–118
34. Mlandenovic N, Hansen P (1997) Variable neighbourhood search. Computers and Operations Research 24(11):1097–1100
35. Palubeckis G (2004) Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Annals of Operations Research 131:259–282
36. Palubeckis G (2006) Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2):279–296
37. Papadimitriou CH, Steiglitz K (1998) Combinatorial Optimization: Algorithms and Complexity. Dover Publications, Mineola, New York
38. Pardalos P, Rodgers GP (1990) Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45:131–144
39. Pardalos P, Xue J (1994) The maximum clique problem. Journal of Global Optimization 4:301–328
40. Phillips AT, Rosen JB (1994) A quadratic assignment formulation of the molecular conformation problem. Journal of Global Optimization 4:229–241
41. Witsgall C (1975) Mathematical methods of site selection for electronic system (ems). NBS Internal Report