# VLR: A Memory-Based Optimization Heuristic

Hansang Yun, Myoung Hoon Ha, and Robert Ian McKay

School of Computer Science and Engineering, Seoul National University
Seoul, 151-744, Republic of Korea

**Abstract.** We suggest a novel memory-based metaheuristic optimization algorithm, VLR, which uses a list of already-visited areas to more effectively search for an optimal solution. We chose the Max-cut problem to test its optimization performance, comparing it with state-of-the-art methods. VLR dominates the previous best-performing heuristics. We also undertake preliminary analysis of the algorithm's parameter space, noting that a larger memory improves performance. VLR was designed as a general-purpose optimization algorithm, so its performance on other problems will be investigated in future.

**Keywords:** Optimization, Metaheuristics, Max-cut problem, memory.

## 1 Introduction

We introduce a novel metaheuristic optimization algorithm which uses a list of already-visited areas (the Visited-Local-Region – VLR) to improve the efficiency of exploration. VLR extends a general local search approach, guiding the system to avoid regions which are unlikely to have favorable solutions or near which we have already searched. The VLR technique is inspired by the biological mechanisms of microRNA, small non-coding RNA molecules which regulates gene expression [1]. The search step size is controlled by a "threshold of uncertainty" (TU), which resembles the temperature in Simulated Annealing (SA). Unlike SA's temperature, TU does not decrease monotonically, but depends on the VLR – we describe it in detail later. We apply VLR to the Maximum Cut (Max-cut) problem with good results.

The remainder of this paper is organized as follows. In Section 2, we introduce previous work on the Max-cut problem. Section 3 describes the VLR heuristic and its application. Section 4 presents the experiment settings for Max-cut, with the computational results being presented in Section 5. We draw conclusions and point out future directions in the last section.

## 2 Background

### 2.1 Search Methods

We propose a stochastic local search algorithm and evaluate it on the Max-cut problem, though it is general in form, and may be useful for other problems. Most

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 151–160, 2014.

stochastic search methods do not employ an explicit long-term memory, either foregoing memory altogether (stochastic hillclimbing and its variants) or relying on an implicit memory either in the population (genetic and swarm algorithms) or in a probability structure (estimation of distribution and ant algorithms). Only a few algorithms, of which tabu search [2] is the most prominent, explicitly remember details of the search space. The algorithm we propose here resembles tabu search in retaining an explicit memory of areas in the search space that it has previously visited. However tabu search uses the memory to determine its neighborhood structure; this algorithm differs in many details, most obviously in using the memory to affect the acceptance criterion rather than the neighborhood structure. It will be described in more detail in the next section.

### 2.2 The Max-Cut Problem

Let $V$ be a set of n vertices, $E$ a set of edges$(i, j)$ with $i, j \in V$, and $W$ a set of weights $w_{ij}$ on the edges $(i, j) \in E$. For a graph $G = (V, E)$, the Max-cut problem seeks a cut$(S, S^c)$ that maximizes the sum of the weights on the edges between $S$ and its complement $S^c$:

$$maxcut(G) = \max_{S \subseteq V}\Big(\sum_{u \in S, v \in S^c} w_{uv}\Big)$$

In spite of its simple conceptualization, this problem is remarkable for its practical applications such as the design of VLSI circuits [3,4], and the determination of ground states of spin-glass models in statistical physics [5]. Since Karp [6] proved it NP-Complete, various algorithms have been used to achieve better solutions with limited computing resources.

Most recent studies focused on heuristic techniques. Burer et al. [7] proposed a rank-2 relaxation heuristic, CirCut, that achieves better solutions than any previous, and can handle bigger problems in shorter computational time. Festa et al. [8] combined a greedy randomized adaptive search procedure (GRASP), variable neighborhood search (VNS), and path relinking (PR) to form a hybrid randomized method. Their VNSPR produces high quality solutions, but with high computational effort (time). These methods can be further hybridized with others, such as CirCut or the Goemans and Willamson algorithm.

The subsequent Scatter Search (SS) heuristic of Martí et al. [9] obtained better performance than CirCut. Kochenberger et al. [10] applied a new Tabu Search algorithm, Diversification-Driven Tabu Search (DDTS) [11], demonstrating its solution quality and computational efficiency on Max-cut problems up to 10,000 vertices. Song and Li's [12] HMA combines a memetic algorithm (MA) with semidefinite programming for initialization, as in [13] GW+Random. It creates a better initial population leading to a higher score for Max-cut than the SS in [9] over many different problems.

We compared our approach with state-of-the-art heuristics, namely SS, HMA, DDTS. Our new method out-performs these older heuristics in most cases.

## 3 Method

Visited-Local-Region (VLR) is a general-purpose metaheuristic optimizing a target function $f(x)$ over solution vectors $x$. In the Max-cut problem, $x$ is a binary vector representing which elements belong to set $S$, and $f(x)$ denotes the value sum for the cut($S$, $S^c$).

We begin with a simpler version, Visited-Local-Hill (VLH), with two main characteristics. It memorizes already-visited local hills, represented by their local optima. Each hill has an attribute, the escape count (EC), proportional to the number of visits (with user-set constant of proportionality $d$). It controls the extent of exploration by a "threshold of uncertainty" $\tau$, differing from SA's temperature in two ways:

1. Its value is reversed (low values mean more exploration).
2. It can both increase and decrease during search.

We give the explanation in three parts: exploration control in the UncertainClimbing method, the VLH heuristic, and the VLR extension of VLH. To simplify the explanation, we limit it to Boolean domains.

### 3.1 Exploration

UncertainClimbing (Algorithm 1) iteratively seeks a local optimum $x$. To permit crossing between hills, we allow it to climb down (the probability varying over time), unlike the random exploration that traditional stochastic hill climbing undertakes when it is stuck in a local optimum. Let $x'$ be a bit-flip neighbor of $x$. We define $h(x)$ and $p(x \to x')$ as:

$$h(x) = max(f(x) - \tau, 0)$$

$$p(x \to x') = \begin{cases} h(x')/(h(x) + h(x')) & , if\,(h(x) + h(x')) > 0 \\ 0 & , otherwise \end{cases}$$

**Algorithm 1.** Pseudocode of UncertainClimbing

```
01: procedure UncertainClimbing(x, τ)
02:   repeat
03:     for i = 1 to N do // N is length of x
04:       let x' be a neighbor of x obtained by flipping the i'th bit of x
05:       pick a random real r between 0 and 1
06:       h(x) = max(f(x) − τ, 0), h(x') = max(f(x') − τ, 0)
07:       if h(x) + h(x') > 0 and r < h(x')/(h(x) + h(x')) then
08:         replace x with x'
09:       end if
10:     end for
11:     increase τ
12:   until (x is a local optimum)
13:   return (x, τ)
14: end procedure
```

**Algorithm 2.** Pseudocode of VLHmain

```
01: procedure VLHmain( )
02:   initialize x with a random solution and τ with f(x) .
03:   repeat
04:     (x, τ) = UncertainClimbing(x, τ)
05:     if x ∉ VLH.keys then enroll x in VLH end if
06:     VLH[x].ec = VLH[x].ec + d
07:     τ = τ − VLH[x].ec
08:   until (end condition)
09:   return the best found solution
10: end procedure
```

$p(x \to x')$ is the probability of accepting $x'$ as the next solution: solutions better than $x$ have higher rates, but worse solutions have some chance of acceptance. The propensity for exploration is managed by $\tau$. For $f(x') \leq \tau$, $p(x \to x') = 0$, so $x'$ is rejected; for $f(x') > \tau$, $p(x \to x') > 0$ and $x'$ may be accepted. Until the process finds the local optimum, $\tau$ increases, decreasing the acceptance rate and making the process more eager, so that it eventually reaches the local optimum.

### 3.2 Visited-Local-Hill (VLH)

RNA silencing works like this: when a messenger RNA contains a genetically vulnerable feature – coding say for diabetes or cancer – microRNAs evolve to silence them by inactivating the matching sequence. VLH acts in a similar way, changing $\tau$ to move the search away from previously matched features.

VLH lists previously visited local optima, implemented as a TRIE structure. This gives match determination time dependent on the search space dimension, but not the list size. When we find a new local optimum not in the VLH list, we add it and initialize the escape count (EC) to a constant $d$. EC is used to decrement $\tau$, determining the acceptance rate for exploration, and thus the distance to the next target. If EC is small relative to the basin of attraction of the local optimum, the process may revisit the same peak, in which case, we increase EC until it is large enough to exit the basin.

Overall, $\tau$ works as follows. When the process is seeking a local optimum, $\tau$ increases; when it reaches one, $\tau$ suffers a large reduction (by EC) and escapes the basin of attraction. Thus the system can act as both a local optimization method and a global one, with $\tau$ and EC adapting the algorithm to the scale of the fitness landscape. Algorithm 2 provides the full details.

### 3.3 Visited-Local-Region(VLR)

VLH is simple and performs competitively. However a small extension can improve it. Searching and escaping from each hill can limit the search scope. We broaden it by defining a local region embracing a few minor hills, represented by the fittest solution, $y$, detected in the region. During this phase, $\tau$ increases – eventually, it will exceed the maximum available $f(x)$ in the region, so search stalls; $\tau$ suffers a reduction by EC and $x$ can move again. VLR is described in detail in Algorithms 3 and 4.

**Algorithm 3.** Pseudocode of VLRmain

```
01: procedure VLRmain( )
02:   initialize x with a random solution and τ with f(x) .
03:   repeat
04:     (x, y, τ) = UncertainClimbing2(x, τ)
05:     if y ∉ VLR.keys then enroll y in VLR end if
06:     VLR[y].ec = VLR[y].ec + d
07:     τ = τ − VLR[y].ec
08:   until (end condition)
09:   return the best found solution
10: end procedure
```

**Algorithm 4.** Pseudocode of UncertainClimbing2

```
01: procedure UncertainClimbing2(x, τ)
02:   y = x
03:   repeat
04:     for i = 1 to N do // N is length of x
05:       let x' be a neighbor of x obtained by flipping the i'th bit of x
06:       pick a random real r between 0 and 1
07:       h(x) = max(f(x) − τ, 0), h(x') = max(f(x') − τ, 0)
08:       if h(x) + h(x') > 0 and r < h(x')/(h(x) + h(x')) then
09:         replace x with x'
10:         if f(x) > f(y) then y = x end if
11:       end if
12:     end for
13:     increase τ
14:   until ( x has been stuck ) // i.e. x is a local optimum and τ ≥ f(x)
15:   return (x, y, τ)
16: end procedure
```

## 4 Experiments

We compared VLR with three leading metaheuristics from section 2: SS [9], HMA [12], and DDTS [10]. SS is the best known algorithm for Max-cut, outperforming well-known methods such as VNSPR [8] and CirCut [7]. HMA has better performance than SS in most cases, but does not dominate. DDTS has even better performance on the same instances, and has been tested on larger problems (3000 to 10000), beyond the range on which SS and HMA were tested. All had been tested on the Gset test set of Helmberg and Rendl [14] of 54 problems, varying from 800 to 3000 vertices, so we used Gset for comparison.

The HMA tests used 30 min. on an Intel Core i5-750 2.67GHz. To calibrate with our Intel Core i7-870 2.93GHz system, we used SPEC2006 benchmarks, giving us a time budget of 27 min. We were able to compare also with SS because [12] provided timings for the same problems under both HMA and SS. [10] does not provide timing details for DDTS, so it is difficult to determine the fairness of the comparison, but we nevertheless provide the comparative attainments of the algorithms.

Ideally, such comparisons should be tested for statistical significance. This was not possible, because the per-run data for previous systems were not available, and the number of runs were too small for statistical stability. To support such comparisons in future, all our tests used 50 runs, and the raw data are available at

http://hdl.handle.net/10371/91260. Suitable parameter settings were determined by some more detailed exploration of VLR's parameter space, which we detail below.

To test the importance of the region search in VLR, we compared VLH and VLR performance on G16, G21, G32, G37, and G52 from Gset. We tested the impact of list size by comparing two extreme cases, VLR lists of sizes 1 and 8,000, on G37. Finally, we tested the sensitivity of VLR's performance to the value of the user-set parameter $d$, trying an exponentially increasing series of sizes – $\sqrt{|V|}/4$,$\sqrt{|V|}/2$, $\sqrt{|V|}$, $2\sqrt{|V|}$, $4\sqrt{|V|}$ – on G16, G21, G37, and G52.

Detailed parameter settings for the experiments are shown in Table 1.

**Table 1.** Experimental Parameter Settings

| Parameter | Value |
|---|---|
| $d$ | $\sqrt{\lvert V\rvert}$ |
| $V$ | Problem dependant |
| list size | $\frac{1\ \text{GByte}}{(\text{size of 1list item})\times\lvert V\rvert}$ |

| Parameter | value |
|---|---|
| the amount of $\tau$ increment | 1 |
| Number of Runs | 50 |

## 5 Result and Discussion

Table 2 shows the results of comparisons between VLR and earlier methods. We show the best known record to date, together with the available data from [12] and [10]. For SS and HMA, we show their best and average performance from 10 runs. Curiously for a stochastic algorithm, the results for DDTS in [10] appear to be from single runs, so that is all we can present. For VLR, we show the best, average and median values from 50 runs, and the success rate.[1]

Bold values indicate best-known performance on a problem. The values enclosed in square brackets are new best-known records. Comparing VLR with SS, VLR has superior performance on 37 problems and worse on one. Comparing with HMA, VLR wins on 30 problems and loses on one. Thus VLR almost dominates SS and HMA; although we cannot statistically test the comparison, it seems that VLR is overall a better performer. The comparison with DDTS is difficult because of the single runs for DDTS, but VLR wins on 27 problems and DDTS on 3. Overall VLR performed well, finding 20 new best-known solutions, while failing to find a known-best solution on only 3 problems. Overall, VLR's performance on Max-cut is clearly of a high order.

Table 3 compares the performance of VLH and VLR. The result of one sample t-test whether all pairwise difference differ from 0 suggests that the region structure brings performance gains on average, though differences are fairly small.

[1] For skewed data such as typically arises in optimization, the median is generally more informative than the mean.

**Table 2.** Comparative Results on Gset Instances

| ID | # of Verts | Best Known | SS Best | SS Avg. | HMA Best | HMA Avg. | DDTS | VLR Best | VLR Avg. | VLR Median | VLR Succ. Rate |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of Runs | | | 10 | | | | 1 | 50 | | | |
| G1 | 800 | **11624** | **11624** | 11624.0 | **11624** | 11624 | **11624** | **11624** | 11622.9 | 11624 | 47 |
| G2 | 800 | **11620** | **11620** | 11620.0 | **11620** | 11620 | **11620** | **11620** | 11620.0 | 11620 | 50 |
| G3 | 800 | **11622** | **11622** | 11619.5 | **11622** | 11622 | 11620 | **11622** | 11622.0 | 11622 | 50 |
| G4 | 800 | **11646** | **11646** | 11638.5 | **11646** | 11646 | **11646** | **11646** | 11646.0 | 11646 | 50 |
| G5 | 800 | **11631** | **11631** | 11630.4 | **11631** | 11631 | **11631** | **11631** | 11631.0 | 11631 | 50 |
| G6 | 800 | **2178** | **2178** | 2174.1 | **2178** | 2178 | **2178** | **2178** | 2178.0 | 2178 | 50 |
| G7 | 800 | **2006** | 1996 | 1988.9 | **2006** | 2006 | **2006** | **2006** | 2006.0 | 2006 | 50 |
| G8 | 800 | **2005** | 1996 | 1994.7 | **2005** | 2005 | **2005** | **2005** | 2005.0 | 2005 | 50 |
| G9 | 800 | **2054** | **2054** | 2051.2 | **2054** | 2053 | **2054** | **2054** | 2054.0 | 2054 | 50 |
| G10 | 800 | **2000** | **2000** | 1999.1 | **2000** | 1999.1 | **2000** | **2000** | 1999.5 | 2000 | 44 |
| G11 | 800 | **564** | **564** | 563.8 | 558 | 558 | **564** | **564** | 564.0 | 564 | 50 |
| G12 | 800 | **556** | 554 | 550.6 | **556** | 552 | **556** | **556** | 556.0 | 556 | 50 |
| G13 | 800 | 580 | 578 | 575.8 | 578 | 578 | 580 | [**582**] | 582.0 | 582 | 50 |
| G14 | 800 | **3063** | **3063** | 3060.8 | 3060 | 3058.1 | 3061 | **3063** | 3061.3 | 3061 | 1 |
| G15 | 800 | **3050** | 3040 | 3036.8 | 3049 | 3048.8 | **3050** | **3050** | 3049.1 | 3049 | 7 |
| G16 | 800 | **3052** | 3044 | 3043.7 | 3050 | 3048.8 | **3052** | **3052** | 3051.0 | 3051 | 10 |
| G17 | 800 | 3046 | 3040 | 3038.4 | 3045 | 3043.6 | 3046 | [**3047**] | 3045.5 | 3046 | 1 |
| G18 | 800 | 991 | 991 | 985.8 | 989 | 986.9 | 991 | [**992**] | 991.3 | 991 | 19 |
| G19 | 800 | **906** | 905 | 898.9 | **906** | 904.1 | 904 | **906** | 905.2 | 906 | 31 |
| G20 | 800 | **941** | **941** | 941.0 | **941** | 941 | **941** | **941** | 941.0 | 941 | 50 |
| G21 | 800 | **931** | **931** | 931.0 | **931** | 930.9 | **931** | **931** | 930.8 | 931 | 47 |
| G22 | 2000 | **13359** | 13349 | 13314.8 | 13358 | 13349.8 | **13359** | **13359** | 13345.8 | 13358 | 9 |
| G23 | 2000 | 13342 | 13323 | 13312.6 | 13337 | 13329.3 | 13342 | [**13344**] | 13335.8 | 13337 | 9 |
| G24 | 2000 | **13337** | 13318 | 13307.8 | 13330 | 13321.8 | **13337** | **13337** | 13322.9 | 13324 | 6 |
| G25 | 2000 | 13332 | 13320 | 13313.8 | 13330 | 13322 | 13332 | [**13335**] | 13325.4 | 13328 | 4 |
| G26 | 2000 | **13328** | 13308 | 13299.7 | 13323 | 13310 | **13328** | 13326 | 13315.8 | 13316 | 8 |
| G27 | 2000 | 3336 | 3332 | 3312.0 | 3334 | 3325.8 | 3336 | [**3341**] | 3326.8 | 3330 | 4 |
| G28 | 2000 | 3295 | 3275 | 3264.9 | 3294 | 3286.9 | 3295 | [**3298**] | 3291.0 | 3294 | 11 |
| G29 | 2000 | 3404 | 3385 | 3376.0 | 3404 | 3386.5 | 3391 | [**3405**] | 3385.8 | 3386 | 4 |
| G30 | 2000 | 3407 | 3395 | 3384.5 | 3407 | 3402.8 | 3403 | [**3412**] | 3402.5 | 3403 | 15 |
| G31 | 2000 | 3305 | 3275 | 3265.8 | 3305 | 3296.3 | 3288 | [**3310**] | 3299.6 | 3301 | 2 |
| G32 | 2000 | 1406 | 1400 | 1393.2 | 1396 | 1392.2 | 1406 | [**1410**] | 1404.4 | 1404 | 1 |
| G33 | 2000 | **1378** | 1364 | 1359.4 | 1372 | 1368.4 | **1378** | **1378** | 1374.5 | 1374 | 3 |
| G34 | 2000 | 1378 | 1368 | 1361.6 | 1378 | 1375 | 1378 | [**1382**] | 1380.1 | 1380 | 15 |
| G35 | 2000 | 7680 | 7654 | 7648.5 | 7680 | 7673 | 7678 | [**7683**] | 7678.9 | 7679 | 2 |
| G36 | 2000 | 7670 | 7667 | 7654.5 | 7670 | 7665.7 | 7670 | [**7675**] | 7671.0 | 7671 | 1 |
| G37 | 2000 | 7682 | 7667 | 7660.3 | 7682 | 7674.6 | 7682 | [**7687**] | 7685.4 | 7686 | 9 |
| G38 | 2000 | 7683 | 7668 | 7659.8 | 7678 | 7669.9 | 7683 | [**7687**] | 7684.2 | 7685 | 3 |
| G39 | 2000 | 2406 | 2395 | 2388.0 | 2406 | 2396.9 | 2397 | [**2408**] | 2399.6 | 2399 | 9 |
| G40 | 2000 | 2393 | 2380 | 2375.9 | 2393 | 2389.2 | 2390 | [**2399**] | 2390.7 | 2394 | 1 |
| G41 | 2000 | **2405** | 2391 | 2385.7 | **2405** | 2401.7 | 2400 | **2405** | 2398.0 | 2405 | 26 |
| G42 | 2000 | 2478 | 2462 | 2458.1 | 2478 | 2469.1 | 2469 | [**2480**] | 2471.0 | 2472 | 2 |
| G43 | 1000 | **6660** | **6660** | 6656.2 | **6660** | 6658.7 | **6660** | **6660** | 6659.1 | 6660 | 48 |
| G44 | 1000 | **6650** | **6650** | 6648.8 | **6650** | 6649.7 | 6639 | **6650** | 66467.0 | 6650 | 30 |
| G45 | 1000 | **6654** | 6646 | 6643.0 | **6654** | 6650.1 | 6652 | **6654** | 6648.7 | 6650 | 16 |
| G46 | 1000 | **6649** | 6647 | 6640.4 | **6649** | 6645.8 | **6649** | **6649** | 6645.4 | 6648 | 18 |
| G47 | 1000 | 6665 | 6655 | 6652.9 | 6656 | 6655.2 | 6665 | 6657 | 6651.3 | 6650 | 9 |
| G48 | 3000 | **6000** | - | - | **6000** | 6000 | **6000** | **6000** | 6000.0 | 6000 | 50 |
| G49 | 3000 | **6000** | - | - | **6000** | 6000 | **6000** | **6000** | 6000.0 | 6000 | 50 |
| G50 | 3000 | **5880** | - | - | **5880** | 5880 | **5880** | 5876 | 5859.6 | 5874 | 15 |
| G51 | 1000 | **3847** | **3843** | 3839.4 | **3847** | 3843.9 | **3847** | **3847** | 3846.0 | 3846 | 9 |
| G52 | 1000 | 3849 | 3841 | 3836.1 | 3848 | 3844.8 | 3849 | [**3851**] | 3849.1 | 3849 | 3 |
| G53 | 1000 | **3849** | 3845 | 3844.3 | **3849** | 3844.9 | 3848 | **3849** | 3846.7 | 3846 | 2 |
| G54 | 1000 | **3851** | 3849 | 3846.0 | 3845 | 3842.5 | **3851** | **3851** | 3850.1 | 3850 | 4 |

**Table 3.** VLH-VLR Performance Comparison

| | VLH Best | VLH Avg. | VLR Best | VLR Avg. | t | p value | mean |
|---|---|---|---|---|---|---|---|
| G16 | 3052 | 3050.9 | 3052 | 3051.06 | 8.593 | <2.2e-16 | 0.16 |
| G21 | 931 | 930.76 | 931 | 930.94 | 11.6283 | <2.2e-16 | 0.18 |
| G32 | 1408 | 1404.88 | 1410 | 1404.84 | -0.6851 | 0.4934 | -0.04 |
| G37 | 7688 | 7683.78 | 7687 | 7684.7 | 12.8641 | <2.2e-16 | 0.92 |
| G52 | 3850 | 3848.74 | 3851 | 3849.08 | 16.462 | <2.2e-16 | 0.34 |

**Table 4.** Performance Comparison for VLR Parameter $d$

| | $\sqrt{\lvert V\rvert}/4$ | $\sqrt{\lvert V\rvert}/2$ | $\sqrt{\lvert V\rvert}$ | $2\sqrt{\lvert V\rvert}$ | $4\sqrt{\lvert V\rvert}$ | t | p-value | mean |
|---|---|---|---|---|---|---|---|---|
| G16 | 3049.1 | **3051.58** | 3051.06 | 3048.6 | 3045.22 | 175.1456 | <2.2e-16 | 6.36 |
| G21 | 921.46 | 929.26 | 930.94 | **931** | 930.32 | 64.8215 | <2.2e-16 | 9.54 |
| G32 | 1392.12 | 1397.6 | **1404.84** | 1400.64 | 1393.16 | 131.3383 | <2.2e-16 | 12.72 |
| G37 | 7677.9 | 7682.62 | **7684.7** | 7678.88 | 7665.5 | 218.2509 | <2.2e-16 | 19.2 |
| G52 | 3847.04 | **3850** | 3849.08 | 3844.88 | 3840.76 | 236.2583 | <2.2e-16 | 9.24 |

Table 4 shows the sensitivity analysis for user-set parameter $d$. We can see that the performance is fairly sensitive to $d$ from the result of one sample t-test for all pairwise difference the best and the worst. Since even small differences in objective values are crucial for optimization performance, setting $d$ correctly is clearly important – however for individual problems, the performance curve seems to be unimodal, with the best settings not varying much. The setting we used in the main experiments ($\sqrt{|V|}$) appears to have been a reasonable

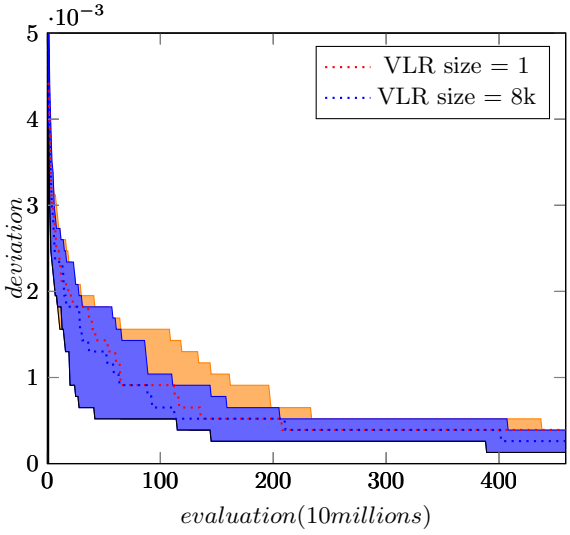


**Fig. 1.** VLR List Size Comparison: Distribution of Deviation Current Best from Best Known Value every $10^7$ Evaluations (Median and Quartiles over 50 Runs) on G37

choice, though a value between $\sqrt{|V|}/2$ and $\sqrt{|V|}$ may have been better. The unimodality of the search means that adaptive measures could be used to choose $d$, but we leave this for future work.

Figure 1 illustrates the effect of VLR list size. The best solutions found each 10 million evaluations from 50 runs on the G37 form the raw data. The vertical axis indicates the deviation from the best known solution. The dotted lines show the median, and the translucent areas indicate the interquartile ranges (rank statistics are more useful here because the distributions are highly skewed). Larger list sizes consistently lead to better performance (though there is substantial overlap). Larger VLR sizes come with a memory cost, but a VLR size of 8,000 is trivial today; because of the use of the TRIE structure, there is little additional time cost. Thus it seems sensible to use reasonably large VLR sizes, even though the performance gains are relatively small. We conclude that the larger list has better performance since the t-test results for all pairwise difference at the end of the runs are $t = 11.9942$, $p - value < 2.2e - 16$, and $mean = 0.064$, respectively.

## 6 Conclusion

The VLH mechanism, inspired by MicroRNA, and extended by the region structure in the VLR algorithm, has shown good performance on the Max-cut problem. VLR's design uses adaptive feedback to maintain the balance between exploration and exploitation through more effective use of information gathered from the search process. We think the performance of VLR derives from its targeted exploration, adapting to the fitness landscape rather than exploring randomly. It thus adopts good features from two important heuristics, namely simulated annealing and Tabu search, combining them in ways that yield useful increments in performance.

In the future we plan to test VLR's performance on a wider range of optimization problems. We also plan to explore the parameter space and algorithm details more deeply – finding ways to determine the optimum list size, and testing whether there are more effective ways to change the EC parameter than the current linear increase and decrease. There is also potential to explore more informed and efficient search operators. Preliminary work in these directions has yielded improved results, but detailed results are not yet available.

**Acknowledgements.** This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Science, ICT & Future Planning(MSIP) / National Research Foundation of Korea (NRF) (Grant NRF-2008-0062609). The ICT at Seoul National University provided research facilities for this study.

## References

1. Chen, K., Rajewsky, N.: The evolution of gene regulation by transcription factors and microRNAs. Nature Reviews Genetics 8(2), 93–103 (2007)
2. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. 13(5), 533–549 (1986)

3. Chang, K.C., Du, D.: Efficient algorithms for layer assignment problem. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 6(1), 67–78 (1987)
4. Pinter, R.Y.: Optimal layer assignment for interconnect. Adv. VLSI Comput. Syst. 1(2), 123–137 (1984)
5. Barahona, F., Grötschel, M., Jünger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Operations Research 36(3), 493–513 (1988)
6. Karp, R.M.: Reducibility among combinatorial problems. In: Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008, pp. 219–241. Springer, Heidelberg (2010)
7. Burer, S., Monteiro, R.D.C., Zhang, Y.: Rank-two relaxation heuristics for max-cut and other binary quadratic programs. SIAM Journal on Optimization 12, 503–521 (2000)
8. Festa, P., Pardalos, P., Resende, M., Ribeiro, C.: Randomized heuristics for the max-cut problem. Optimization Methods and Software 17(6), 1033–1058 (2002)
9. Martí, R., Duarte, A., Laguna, M.: Advanced scatter search for the max-cut problem. INFORMS J. on Computing 21(1), 26–38 (2009)
10. Kochenberger, G.A., Hao, J.K., Lü, Z., Wang, H., Glover, F.: Solving large scale max cut problems via tabu search. Journal of Heuristics 19(4), 565–571 (2013)
11. Glover, F., Lü, Z., Hao, J.K.: Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR, Q. J. Oper. Res. 8(3), 239–253 (2010)
12. Song, B., Li, V.: A hybridization between memetic algorithm and semidefinite relaxation for the max-cut problem. In: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference, GECCO 2012, pp. 425–432. ACM, New York (2012)
13. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM 42(6), 1115–1145 (1995)
14. Helmberg, C., Rendl, F.: A spectral bundle method for semidefinite programming. SIAM Journal on Optimization 10, 673–696 (1997)