Engineering Stochastic Local Search for the Low Autocorrelation Binary Sequence Problem

Steven Halim, Roland H.C. Yap, and Felix Halim

School of Computing, National University of Singapore
{stevenha,ryap,halim}@comp.nus.edu.sg

Abstract. This paper engineers a new state-of-the-art Stochastic Local Search (SLS) for the Low Autocorrelation Binary Sequence (LABS) problem. The new SLS solver is obtained with white-box visualization to get insights on how an SLS can be effective for LABS; implementation improvements; and black-box parameter tuning.

1 Introduction

Low Autocorrelation Binary Sequence (LABS) problem is a hard problem with simple formulation: find a binary sequence $s = \{s_0, s_1, \ldots, s_{n-1}\}, s_i \in \{-1, 1\}$ of length n that minimizes the objective function $E(s)$ – the *quadratic* sum of the autocorrelation function C_k, or equivalently, maximizes the merit factor $F(s)$:

$$C_k(s) = \sum_{i=0}^{n-k-1} s_i s_{i+k} \qquad E(s) = \sum_{k=1}^{n-1} (C_k(s))^2 \qquad F(s) = \frac{n^2}{2E(s)}$$

The LABS problem dates from 1960s and was first posed in the Physics community. It has applications in many communication and electrical engineering problems. More recently, LABS has been investigated by the optimization community using both exact and incomplete solvers [1,2,3,4,5].

In 2006, Dotú and van Hentenryck [4] proposed a *simple* SLS: Tabu Search (TS) *with frequent restarts*. This could find optimal LABS solutions for $n \leq 48$ much quicker than the exact Branch & Bound [2]. It was roughly on par with another good SLS solver for LABS problem: Kernighan-Lin [3].

In 2007, Gallardo *et al.* [5] proposed an SLS: MA$_{TS}$, combining a Memetic Algorithm with a similar TS. MA$_{TS}$ was shown to be "one order of magnitude" faster than the *pure* TS [4] and was the fastest LABS solver in 2007.

In this paper, we show how an integrated white+black box approach [6] using an SLS engineering tool Viz [7,8] can be used to successfully engineer a new state-of-the-art LABS SLS starting from [4]. For more details, please visit `http://sls.visualization.googlepages.com`.

2 In-Depth Analysis of LABS Fitness Landscape

Previous researchers, e.g. [1,2,5] have shown several features of LABS fitness landscape. As LABS is unconstrained, each LABS instance n has 2^n valid solutions with several Global Optima (GO) (≥ 4) (an approximation of $|GO|$ for

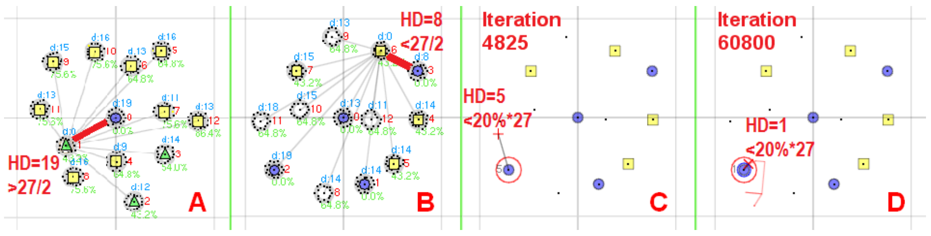

Fig. 1. FLST visualization for LABS $n = 27$ with 4 GO (dark **blue** circles). LO are deep and isolated, shown by black dots (poor solutions) around each LO in part A&B.

$3 \leq n \leq 64$ is in [3]). These GO are spread like 'golf holes' (deep and isolated) in irregular LABS fitness landscape. The fitness landscape of LABS causes difficulties for standard SLS algorithms to work well especially with large n.

In order to get more insights about LABS fitness landscape, we use the Fitness Landscape Search Trajectory (FLST) visualization in Viz [7,8]. To obtain this FLST visualization, we run *our* initial implementation (called **TSv1**) of the TS algorithm from [4] to sample diverse and high quality Local Optima (LO) from the fitness landscape of medium-sized LABS instances ($n \leq 40$). Our sampling strategy exploits the symmetries in LABS: when **TSv1** reaches a solution with Objective Value (OV) equals with the known optimal value (a GO) for that particular medium-sized LABS instance, we can immediately generate all the symmetries of this GO solution. This sampling strategy is used to get a clearer picture of the LABS fitness landscape (compare Fig. 1.A with 1.B).

In Fig. 1.A, we see that without symmetry in GO/LO sampling, we are not immediately aware of the existence of other GO and the *Hamming Distance* (HD) from the current LO to the nearest GO found seems to be large, HD $> n/2$.

By exploiting symmetry, all 4 GO are also 'found' when **TSv1** hits a GO. In Fig. 1.B, we can now see that the positions of GO (dark **blue** circles) are spread out. This suggests that wherever the current solution is, it should be nearer to one GO (the *nearest GO*) than to other GOs. Further observations reveal that LO (light colored non-**blue** circles) are usually not too close to the nearest GO. By using exact enumeration for LABS $3 \leq n \leq 24$, we have checked that around 85% of the 2nd best solution (which is an LO) have HD around $[n/4 \ldots 2n/5]$ bits away from the nearest GO. We exploit this insight.

3 Improving the Tabu Search Algorithm of [4]

In [4], a rather simple yet successful TS algorithm for LABS is presented. We are grateful to the TS code (we call this **TSv0**) from the authors. We benchmarked **TSv0** on our test machine, a 2 GHz Centrino Duo laptop (see the scattered **black** circles around **magenta** line in Fig. 2). Benchmarking shows that our test machine has similar performance to the 3 GHz P4 PC used in [4].

Before obtaining the **TSv0** code, we implemented the TS algorithm using *our own* understanding of the pseudo-code in the paper. We call our original

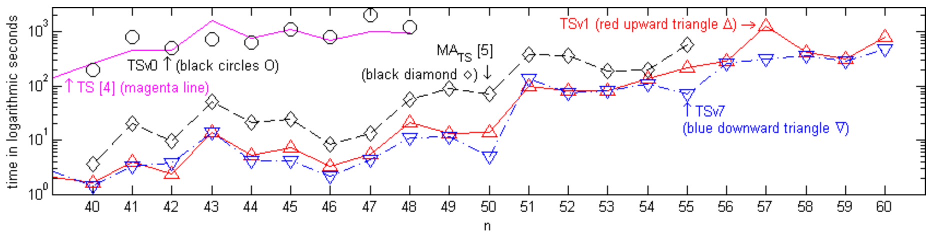

Fig. 2. Comparison of average runtimes (20 runs) between {**TS [4]**, **TSv0 O**}, {**MA**$_{TS}$ **[5]** ◇}, and {**TSv1** △, **TSv7** ▽} for LABS with known optimal OVs ($40 \leq n \leq 60$)

implementation, **TSv1**. **TSv1** is already much faster by about one order of magnitude than **TSv0** (see the **red** line with upward triangle in Fig. 2).

Visualization in Viz shows clearly the speed difference as the search trajectory in **TSv1** animates much faster than **TSv0**. Analysis of the source codes reveals the following two major differences. First, while both codes use a form of "incremental computation" to speed up the naïve $O(n^2)$ $E(s)$ computation, the actual sub-algorithms for this part turns out to be different. Since this part is not described in [4], we implemented **TSv1** with the incremental $O(n)$ *ValueFlip* technique used by MA$_{TS}$ [5]. It turns out that although there is some incremental calculation in **TSv0**, the computation of $E(s)$ is still $O(n^2)$.

Second, though both codes use an $O(1)$ tabu **table** mechanism, they have different `TABU_TENURE` settings. We know that `TABU_TENURE` cannot be $\approx n$ as it will quickly forbid (almost) all 1-bit flip moves. Black-box tuning on several constant values $[0.1, 0.2, 0.3]n$ on training instances $n = \{27, 30, 41, 42\}$ helps us to set small `TABU_TENURE` $= 0.2n$ for **TSv1**. But, **TSv0** use `TABU_TENURE` $= n$. Thus **TSv0** does more frequent random restart (every n+1 iterations) than the pre-determined `MAX_STABLE` parameter as no more valid moves are available.

We can see that **TSv1** runtimes are already comparable to the recent state-of-the-art MA$_{TS}$ (**TSv1** on 3 GHz P4 PC is about 1.7 to 5.6 times faster than MA$_{TS}$ [5] for LABS $40 \leq n \leq 55$ and the 3 GHz P4 PC is (probably) at most 1.25 times faster than the 2.4 GHz P4 PC in [5]). We remark that this shows that the random restart strategy in **TSv0/TSv1** is good and better than the benchmarking in [5] would indicate.

4 A State-of-the-Art Tabu Search for LABS

We wanted to get better results. We analyzed **TSv1** search trajectory using the same FLST visualization. During visualization, a circle is drawn on the nearest sampled GO/LO if the current solution is "near" (near is HD $\leq 20\% * n$).

Using this feature, we observe the following behavior, shown in Fig. 1.C and Fig. 1.D: **TSv1** happens to be near a GO in the *earlier phase* of the search (1.C), but **TSv1** does not immediately navigate there. **TSv1** then wanders to another region near to another GO, perhaps due to random restart strategy. *Thousands* *of iterations later*, **TSv1** gets near to the first GO again (1.D) and this time **TSv1** manages to find the GO.

Such observations and insights about the LABS fitness landscape in Sec. 2 lead us to engineer a better SLS strategy. The resulting TS variant is called **TSv7**. (We experimented with other variants also using the white+black box approach described here but **TSv7** had the best performance). Upon experiencing stagnation, **TSv7** restarts to a region around HD $n/4$ bits away from the current LO. Basically, **TSv7** searches for the *nearest* GO. Only if the current LO region is saturated, **TSv7** does a diversification.

The implementation improvements using faster incremental calculation, improved `TABU_TENURE` settings and the new strategy engineered from white-box analysis using visualization gives a new SLS, **TSv7**. However, we are not done. To obtain a state-of-the-art result, we configured the parameter values for **TSv7** using black-box tuning. We ran a full factorial design of logical parameter values on a set of training instance and picked the best one. Due to space constraints, we are unable to show the algorithm and its parameters. More details and the source code for **TSv7** can be found on the webpage.

Fig. 2 shows the performance of TS (timings from [4]), MA_{TS} (timings from [5]), and **TSv0/TSv1/TSv7** (2 GHz Centrino Duo laptop). We see that **TSv7** strategy is better than the original random restart strategy used in **TSv0/TSv1**. The performance gap is easily noticeable on larger $n = \{50, 55, 57, 60\}$.

To analyze the results, we used the Wilcoxon signed-ranks test. It detected a significant difference between the average runtimes of **TSv1** and **TSv7** on LABS $40 \leq n \leq 60$ (21 pairs, $T = 27.5, p < .01$). Since both TS variants use the same

Table 1. Best found LABS solutions using **TSv7**: $61 \leq n \leq 77$. These runs are performed on a 2.33 GHz Core2 Duo PC.

n	E(s)	F(s)	Runtime	Limit	Best Found LABS in Run Length Notation [2]
61	226	8.23	3 m	1.1 h	332111121112351831212221111311311
62	235	8.18	8 m	1.5 h	112212212711111511121143111422321
63	207	9.59	4 m	2.0 h	221222115121145111711111 2323231
64	208	9.85	47 m	2.7 h	2232241113411211151111172122 12212
65	240	8.80	2.2 h	3.7 h	1323232111117111541121511222 12211
66	265	8.22	3.1 h	4.9 h	2432112312311211212412318111111 1311
67	241	9.31	4.1 h	6.6 h	12112111211222B2221111111112224542
68	250	9.25	6.6 h	8.8 h	11111111141114723212325141211 2221212
69	274	8.69	8.2 h	11.8 h	111111111114111472321232514121 12221212
70	295	8.31	12.4 h	15.8 h	23244121117222141611252123111 11111
71	275	9.17	7.8 h	10.0 h	24124412417222211111311231121 1231121
72	300	8.64	2.4 h	10.0 h	1111114111444171151122142122224222
73	308	8.65	1.2 h	10.0 h	11111123112311221131112121141713 22374
74	349	7.85	0.2 h	10.0 h	1132132161233312511141212112251113 1111
75	341	8.25	8.0 h	10.0 h	12122132121211211111131111618433213232
76	338	8.54	4.6 h	10.0 h	111211112234322111134114212211221311B11
77	366	8.10	3.9 h	10.0 h	1111111913422224311233122134112121 12112

incremental OV computation and run on the same hardware, this difference in average runtimes can be attributed to the new stochastic strategy.

The least square fit on the logarithm of the average runtimes gives an estimated running time of $O(5.03e\text{-}6 * 1.37^n)$ and $O(1.03e\text{-}5 * 1.34^n)$ seconds for **TSv1** and **TSv7**, respectively. We believe **TSv7** to be the *current* state-of-the-art SLS algorithm for LABS. Due to lack of space, we do not show the error bars but the experiments show that **TSv7** is more robust than MA_{TS} and **TSv1**.

Table 1 explores the frontier of LABS instances, $61 \leq n \leq 77$, where optimal values have yet to be proven. For $61 \leq n \leq 70$, we use a runtime limit roughly based on the estimated runtime from Fig. 2. For $n > 70$, we use a runtime limit of 10 hours. We see that **TSv7** manages to obtain relatively good LABS solutions (by the merit factor) in reasonable running time.

5 Conclusion

The contributions of this paper are twofold. First, we show that one has to analyse the search trajectory and not just the timings for a SLS. Our implementation (**TSv1**) of the pseudo-code in [4] shows that it is actually a good strategy. The conclusion in [5] that MA_{TS} is faster than the original TS by 'one order of magnitude' is in part due to the less incremental implementation.

Second, we have shown how to engineer a new state-of-the-art LABS solver. Though the changes from **TSv1** to the final **TSv7** may seem small, it is often that small changes to an SLS causes big differences. Our changes are derived from reasoning on the LABS fitness landscape structure and TS trajectory behavior and thus serve as a rationale supported by empirical experiments. The resulting **TSv7** is also simpler than the hybrid MA_{TS} code.

Acknowledgements

We thank Iván Dotú and Pascal van Hentenryck for sharing their source code.

References

1. CSPLIB: A Problem Library for Constraints, `http://www.csplib.org`
2. Mertens, S.: Exhaustive search for low-autocorrelation binary sequences. Journal of Physics A: Mathematical and General 29, 473–481 (1996)
3. Brglez, F., Xiao, Y.L., Stallmann, M.F., Militzer, B.: Reliable Cost Predictions for Finding Optimal Solutions to LABS Problem: Evolutionary and Alternative Algorithms. In: Intl. Workshop on Frontiers in Evolutionary Algorithms (2003)
4. Dotú, I., van Hentenryck, P.: A Note on Low Autocorrelation Binary Sequences. In: Constraint Programming, pp. 685–689 (2006)
5. Gallardo, J.E., Cotta, C., Fernández, A.J.: A Memetic Algorithm for the Low Autocorrelation Binary Sequence Problem. In: GECCO, pp. 1226–1233 (2007)

6. Halim, S., Yap, R.H.C., Lau, H.C.: An Integrated White+Black Box Approach for Designing and Tuning SLS. In: Constraint Programming, pp. 332–347 (2007)
7. Halim, S., Yap, R.H.C., Lau, H.C.: Viz: A Visual Analysis Suite for Explaining Local Search Behavior. In: ACM User Interface Software & Technology, pp. 57–66 (2006)
8. Halim, S., Yap, R.H.C.: Designing and Tuning SLS through Animation and Graphics: an Extended Walk-through. In: Engineering SLS Algorithms, pp. 16–30 (2007)