

Optimizing a Discrete Switching Pattern Using Two Simulated Annealing Algorithms

Mohammad Tavakoli Bina
University of Surrey
m.tavakoli@eim.surrey.ac.uk

David C. Hamill
University of Surrey
d.hamill@surrey.ac.uk

Abstract — In many applications, the output waveform of an inverter must be sinusoidal. However, the output also contains low and high order harmonics due to switching. As an alternative to analog PWM, a digital basis can be established for the switching sequence. The desired sinusoidal period is divided into N intervals, each taking the value '+1' or '-1' to form a bit-string. In this paper, Simulated Annealing (SA) is employed to choose a bit-string which optimizes the harmonic performance, subject to constraints. The method is applied to a new FACTS controller, the Bootstrap Variable inductance (BVI). Next, to obviate the complicated and slow SA algorithm, a new Simplified Simulated Annealing (SSA) algorithm is developed and applied to the BVI. Finally, the harmonic structures of the switching sequences generated by SA, SSA, and PWM are compared.

Keywords - Simulated Annealing, optimization, switching sequence, Bootstrap Variable Inductance, FACTS.

I. Introduction

A. PWM Schemes

PWM schemes, with a sinusoidal reference and a ramp (or triangle) carrier, generate an output which contains almost no low-order harmonics. The higher order harmonics can be filtered more easily than the lower harmonics. To achieve lower total harmonic distortion (THD), lower ripple, and more accurate fundamental magnitude, the carrier frequency should be as high as possible.

Whatever the carrier frequency, the harmonic at that frequency is strong. However, the lower the modulation index (the ratio of the reference magnitude to the carrier magnitude), the stronger the carrier frequency harmonic. Moreover, at any given modulation index, there is a critical carrier frequency above which the magnitude of the carrier frequency harmonic stays constant. To show this, consider an inverter driving an inductance. A sinusoidal PWM scheme controls the switching sequence. We define the *gain* of the system as the ratio of the magnitude of the fundamental component of the PWM output to the magnitude of its reference input. A MATLAB program was written to simulate the inverter output voltage and

the inductor current for different carrier frequencies. Fig. 1(a) shows the magnitude of the carrier frequency harmonic versus the carrier frequency for different gains. For a gain of 2, the carrier frequency harmonic varies from 0.35 p.u. with a 100 Hz carrier to 0.42 p.u. with a 350 Hz carrier. Although the carrier frequency harmonic remains constant above 350 Hz, it is still desirable to use a higher frequency in order to eliminate the low order harmonics.

Fig. 1(b) shows the simulated fundamental current of the inductor versus the carrier frequency for a gain of 1.6 (modulation index 0.8). The current coincides with its expected theoretical value of 0.16 only for carrier frequencies above 450 Hz. This is a third reason for using a high carrier frequency.

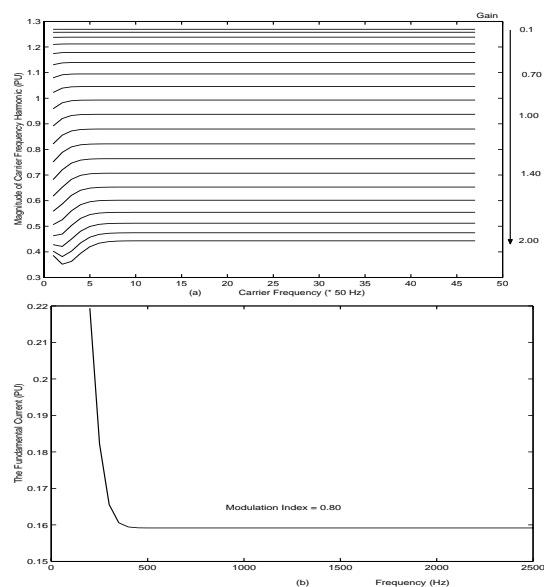


Figure 1: (a) Magnitude of the carrier frequency harmonic versus the carrier frequency. The parameter is the gain; the top curve corresponds to a gain of 0.1 and the lowest to a gain of 2. (b) The fundamental current versus the carrier frequency.

On the other hand, the higher the carrier frequency, the larger the number of switching transitions, increasing the switching losses. Therefore we have to make a design tradeoff: we want to as high a carrier frequency as possible, subject to obtaining adequately high efficiency. This leads to the formulation of the practical design problem in terms of minimizing an objective function involving the harmonic

magnitudes and some measure of switching loss.

B. Bit string scheme

It is also possible to find a switching sequence in a digital environment which satisfies our defined objectives. For example, the synchronous period of power systems, say 20ms, can be divided into N intervals each of duration $20/N$ ms. Each interval is associated with a digit. This sequence of digits forms a string which controls the switches of an inverter. The string can be either a bi-level sequence or a tri-level sequence. For the bi-level case, the digits are ‘bits’ taking values of ‘+1’ or ‘-1’. For a tri-level sequence the value of a digit is ‘+1’, ‘0’ or ‘-1’. Bi-level and tri-level sequences can both be produced by full-bridge inverter circuits. Now, the quantity and positions of ‘+1’s, ‘0’s and ‘-1’s can be selected so that the objective function is minimized, without the restriction of a fixed carrier frequency as in PWM.

A general optimization problem consists of an objective function (expressing our goals) which should be minimized subject to a set of constraints. Considering the space of N -bit strings, there are 2^N possible sequences; for tri-level sequences, 3^N . Among these, a sequence should be sought for which specific objectives are satisfied subject to a number of constraints. The real problem is that the solution space expands exponentially with N : e.g. for $N = 256$, $2^N = 1.16 \times 10^{77}$ and $3^N = 1.39 \times 10^{122}$. In order to find a global minimum of the objective function, an exhaustive search of such a large solution space is impractical. But methods based on local minimization run the risk of missing the global minimum. We therefore desire a technique that is both reliable and fast: it should approach the global minimum with a small number of objective function evaluations.

Several random-based techniques have been developed to search large spaces, including genetic algorithms and simulated annealing. We concentrate here on simulated annealing and a new simplified method, and give an example concerning a recently proposed FACTS controller, the Bootstrap Variable Inductance (BVI) [1]–[4]. Henceforth, we confine our attention to bit strings (bi-level sequences).

II. Simulated Annealing

Annealing is the physical process of heating up a solid (such as iron) until it melts, followed by cooling it down until it crystalizes into a state with a perfect lattice [5]–[6]. During the process of annealing the free energy of the solid decreases, and may be minimized, depending on the cooling down process. In a combinatorial optimization, a similar process can be established. In this analogy, the objective function corresponds to the free energy and the feasible solutions to the physical state. The resulting method is called *Simulated Annealing* (SA).

The SA algorithm uses the Metropolis acceptance procedure, as follows. Let T be the thermal equilibrium (temperature) and s_i be the configuration vector of the system. The probability $P_T(s_i)$ that s_i is the equilibrium point depends on the system energy $E(s_i)$ for this vector, and follows the Boltzmann distribution

$$P_T(s_i) = \frac{e^{-\frac{E(s_i)}{kT}}}{\sum_n e^{-\frac{E(s_n)}{kT}}} \quad (1)$$

where k is Boltzmann’s constant. To move from the present vector s_i , a new configuration vector s_j is randomly found. The probability ratio for the two system configuration vectors is:

$$Q_T(s_j) = \frac{P_T(s_j)}{P_T(s_i)} = e^{-\frac{E(s_j) - E(s_i)}{kT}} \quad (2)$$

The Metropolis acceptance procedure states if $\Delta E = E(s_j) - E(s_i) < 0$, then the new configuration vector s_j is automatically accepted; if not, it is accepted with a probability of $Q_T(s_j)$. A specific value of $Q_T(s_j)$ can be generated as a random number between zero and one. It has been proved [5] that the Metropolis acceptance procedure, combined with the Boltzmann distribution, will converge asymptotically to the global optimum solution provided that the cooling down is sufficiently slow. The Metropolis acceptance procedure is shown in Fig. 2. (However, there is no indication of what is meant by “sufficiently” slow!) The

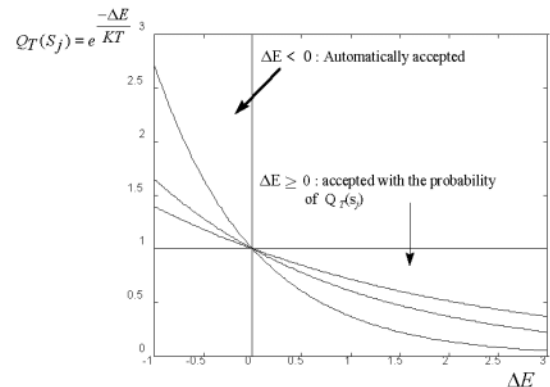


Figure 2: The Metropolis acceptance procedure based on the Boltzmann distribution.

SA algorithm consists of two parts. First, for a given temperature T , a sufficient number of new solutions should be randomly generated. For every new solution, the objective function is evaluated and assessed with the Metropolis acceptance procedure. Once an acceptable solution is found, future random searches take place around it until a better solution is found. This is like a local search except the acceptance procedure can jump out of a local minimum. The second part of the SA algorithm is a repetitive heating and cooling procedure, which manages the first part, changing its temperature T .

In [5], some suggestions are made for initializing T and for the cooling-down step size, ΔT . However, it is not

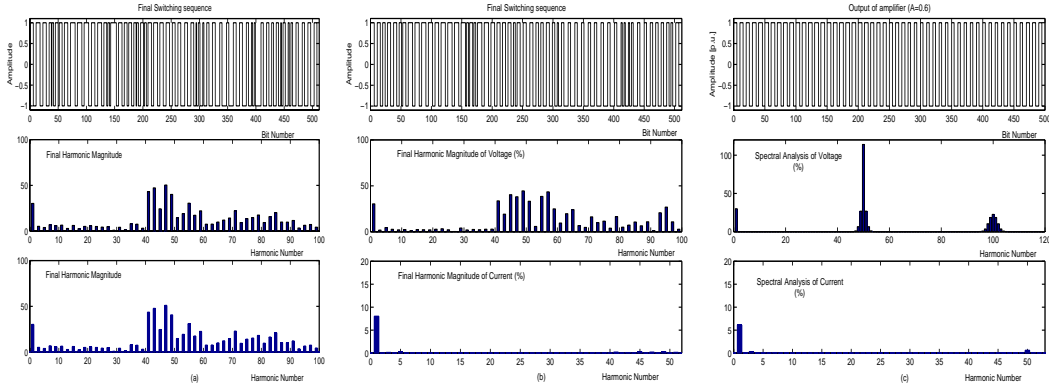


Figure 3: The switching sequence (a) optimized with SA, (b) optimized with SSA, and (c) with sinusoidal PWM (2500 Hz carrier)

obvious how to select the best initial values for every temperature schedule, for ΔT , and for the “melting point.”

A. Simplified Simulated Annealing (SSA) algorithm

This paper introduces an alternative algorithm, *Simplified Simulated Annealing* (SSA). (We are not aware of any previous publications, but the algorithm is so straightforward that it may well have been thought of before.) SSA is basically similar to SA, but it is simpler to implement and understand. Instead of the Metropolis procedure, SSA only accepts improved solutions ($\Delta E < 0$), and uses a different random process from SA. Unlike SA, a local minimization procedure accepts only improved solutions. However, the SSA acceptance procedure is not completely local, as the improved solutions are stored in a temporary variable and search continues around the original point, retaining the possibility of finding a better, non-local minimum.

The algorithm starts from an initial vector s_0 , chosen at random, and searches for better solutions by randomly changing arbitrary bits of s_0 . The best of the accepted solutions is chosen to become the new initial vector, s_1 . The bit-changing procedure is repeated for s_1 but with a narrower range: fewer bits are changed. The process repeats until the range of vector variation is very small, just a few bits, then terminates.

For example, if $N = 256$ we might start by varying from 1 to 127 bits of the initial bit-string, the number being chosen at random. We might do this for say 100 attempts. At the end of the procedure, we might end up by varying only one bit of the final vector at a time (again for 100 attempts). Therefore, only two parameters have to be set up for SSA: first, the number of searches to be done around each initial vector (100 in this example); and second, the maximum value of the initial vector variation (127 bits here). We have not analyzed the performance of this algorithm, but despite its simplicity, it seems robust and efficient.

Appendix A contains a sample C program implementing the SSA algorithm. The function `evaluate` must be provided by the user.

III. Application to BVI

The Bootstrap Variable Inductance (BVI) is a recently proposed FACTS controller that emulates variable positive and negative inductance [1]–[4]. As far as the present paper is concerned, it may be considered as a sinusoidal inverter and an inductance.

Shaw et al. [7] studied an inverter and presented a MATLAB program which searches for an optimal 1024-bit tri-level switching sequence that approximates a sine wave with unity amplitude. In their program the number of ‘+1’s is kept constant throughout, based on the average of a tri-level sequence in a half-cycle of sine wave. (We are unable to establish a theoretical basis for this assumption, however.) We have extended this program for a bi-level sequence, while the number of ‘+1’s can be varied freely. Also, the harmonic analysis required by the objective function is based on a recently proposed analytical method [2], [3], which is much faster than MATLAB’s internal FFT function.

The SA and SSA algorithms were applied to the BVI’s switching function, a bi-level bit-string. As the BVI is connected to a power system, the desired output frequency is 50Hz. We define a 256-bit string over $[0, 0.01]$ seconds, making use of half-wave symmetry. The aim is to find a bit-string which satisfies these requirements:

- A small number of switching transitions (for low switching loss).
- Insignificant levels of low order harmonics.
- No dominant high order harmonics.

The last of these is achieved by forcing the high order harmonics to have approximately equal magnitudes, preventing a single frequency from dominating.

These goals have been formulated as an objective function. The optimization problem also is subject to some constraints. For example, the fundamental magnitude is to be 0.3 p.u. (This value was chosen because it leads to

Table 1: Comparison of SA, SAA, and PWM

Algorithm	Fundamental		Total Harmonic Percentage	Number of '+1's	Switching Transitions	Maximum Harmonic	
	Magnitude	Phase Angle				Number	Value
SA	0.3024	-1.5706	121.0	155	110	47	0.506
SSA	0.3028	-1.5903	123.5	155	109	49	0.425
PWM	0.3000	-1.5708	132.7	153	99	50	1.139

potentially large high order harmonics.) Each time a candidate vector s_j is randomly generated, the corresponding fundamental magnitude is calculated. If it differs greatly from 0.3 p.u., a heavy penalty is added to the objective function. Similarly, deviations of the fundamental phase angle from its intended value of $-\pi/2$ are also penalized. We set the number of 'low order' harmonics to 40, adding a further penalty term for large amplitudes of the first 40 harmonics.

IV. Optimization Results

Fig. 3 summarizes the results graphically for the SA and SSA optimizations, with PWM for reference.

Fig. 3(a) shows the result of the SA algorithm. The starting point s_0 was derived from an analog PWM waveform. Empirically, we settled on 11 temperature schemes, 200 temperature decrements, and 256 random vectors at every temperature. In other words, the maximum number of function evaluations is 563,200; however, the actual number could be less, depending on luck. The top waveform is the optimized bi-level switching sequence which contains 110 switching transitions and 155 '+1's in a half-cycle (10ms). The middle graph is its spectrum based on the analytical method proposed in [3]. The fundamental magnitude is 0.3024 p.u. and the phase angle is -1.5706 rad. The bottom waveform is the spectrum using MATLAB's internal FFT function, for comparison.

The SSA algorithm was also applied to the optimization problem and its result can be found in Fig. 3(b). Unlike the SA algorithm, the starting point was a random vector. The number of bits changed in each iteration started at 148 bits and finished at 1 bit (out of 256). At each of these 148 stages, 500 function evaluations were performed, giving a fixed total of 74,000 iterations. (Unlike SA, this number is predetermined.) The fundamental magnitude is 0.3070 p.u. and the phase angle is -1.5903 . Again the top waveform is the switching sequence and the middle graph is the spectrum, but now the bottom graph is the spectrum of the inductor current.

Both SA and SSA achieve a good performance in the frequency domain, in line with the objectives: the fundamental magnitude and phase are very close to their desired values; the low order harmonics (≤ 40) are of very low level; and the high order harmonics (> 40) are evenly spread, with no dominant spectral line. In comparison, the PWM waveform, Fig. 3(c), shows very small low order harmonics,

but a large component at the carrier frequency (50th harmonic). In fact, the 50th harmonic is nearly four times as large as the fundamental, and would be difficult to filter out.

Table 1 compares the switching sequences optimized by SA and SSA (discrete patterns), and also the sinusoidal PWM scheme (analog pattern — a 2500Hz triangular carrier as in Fig. 3(c)). SA and SSA approach an acceptable solution. However, the level of maximum harmonic magnitude is lower in SSA whereas the total harmonic percentage is lower in SA. In other words, the spectrum of the SSA-optimized sequence is more uniform with a lower level than with SA.

The SA and SSA results are comparable. However, the SA optimization started from a good initial point (the PWM waveform) and needed up to 563,200 evaluations depending on the random number sequence, while the SSA optimization started from a random initial point and needed only 74,000 evaluations. (This is only $1/1.56 \times 10^{72}$ of the solution space!) In our estimation, this makes SSA a superior algorithm, at least for this particular problem. Moreover, it is easier to understand, program and select parameter values for SSA than for SA.

V. Conclusion

The bit-string switching sequence approach has been discussed and applied to the BVI FACTS controller. In comparison with PWM, it has several advantages but also some disadvantages. Following a discussion of Simulated Annealing, a Simplified Simulated Annealing algorithm has been introduced. It is not only simpler to understand and apply than SA, but also it finds comparable solutions in less time. However, it is not proved to converge to a global optimum. Although no analysis has been performed, SSA seems more efficient than SA while retaining its robustness.

References

- [1] D.C. Hamill and M. Tavakoli Bina, "The Bootstrap Variable Inductance and its applications in AC power systems", *IEEE Applied Power Electronics Conf.*, vol. 2, pp. 896–902, March 1999
- [2] M. Tavakoli Bina and D.C. Hamill, "The Bootstrap Variable Inductance: a new FACTS control ele-

ment”, *IEEE Power Electronics Specialists Conf.*, vol. 2, pp. 619–625, June 1999

- [3] M. Tavakoli Bina and D.C. Hamill, “Harmonic analysis of the PWM series/parallel bootstrap variable inductance”, *IEEE Power Electronics and Drive Systems*, vol. 1, pp. 22–27, July 1999
- [4] M. Tavakoli Bina and D.C. Hamill, “Average model of the Bootstrap Variable Inductance” , *IEEE Power Electronics Specialist Conference*, June 2000
- [5] E.H.L. Aarts, “Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing”, Wiley 1997
- [6] L. Davis, “Genetic algorithms and simulated annealing”, Pitman 1987
- [7] S.R. Shaw, D.K. Jackson, T.A. Denison and S.B. Leeb, “Computer aided design and application of sinusoidal switching patterns”, *COMPEL’98 Record*, Como, pp. 185–191

Appendix A – The SSA Algorithm

```
/* Simplified Simulated Annealing algorithm in ANSI C */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
/* The number of bits in the bit strings: */
#define BITS 512
```

```
/* The number of outer iterations: */
#define I_MAX 150
```

```
/* The number of inner iterations: */
#define J_MAX 50
```

```
struct Candidate
{
    int string[BITS]; /* a bit string */
    double badness; /* its badness (to be minimized) */
};
```

```
void evaluate (struct Candidate *pc);
```

```
/*
    User-supplied objective function to evaluate a bit
    string and assign a "badness" value, which the
    algorithm will try to minimize
*/
```

```
void main (void)
```

```
{
    int i;
    struct Candidate best;
```

```
/* Initialise "best" to a random value: */
```

```
srand (time (NULL));
for (i = 0; i < BITS; i++)
    best.string[i] = (rand () > RAND_MAX / 2) ? 1 : 0;
evaluate (&best);
```

```
/* Outer loop: "decreasing temperature": */
```

```
for (i = I_MAX; i > 0; i--)
{
    struct Candidate save = best;
    int j;
```

```
/* Inner loop: search around "best": */
```

```
for (j = 0; j < J_MAX; j++)
{
    int k;
    struct Candidate try = best;
```

```
/* Randomly change i bits of the candidate string: */
```

```
for (k = 0; k < i; k++)
{
    int b;
```

```
/* Pick a bit at random and flip it: */
```

```
b = (int)(rand () * BITS / RAND_MAX);
try.string[b] = 1 - try.string[b];
}
```

```
/* Evaluate this candidate; if good save it: */
```

```
evaluate (&try);
if (try.badness < save.badness)
{
    save = try;
```

```
} /* End of inner loop */
```

```
best = save; /* the best candidate so far */
```

```
} /* End of outer loop */
```

```
puts ("Best candidate found:");
printf ("string = ");
```

```
for (i = BITS - 1; i >= 0; i--)
    printf ("%i", pc->string[i]);
```

```
printf ("; badness = %f\n", pc->badness);
```

```
}
```