An Adaptive Memory Procedure for Continuous Optimization

Abraham Duarte
Universidad Rey Juan Carlos
abraham.duarte@urjc.es

Rafael Martí
Universidad de Valencia
rafael.marti@uv.es

Fred Glover
OptTek Inc.
glover@opttek.com

Abstract

In this paper we consider the problem of finding a global optimum of an unconstrained multimodal function within the framework of adaptive memory programming, focusing on an integration of the Scatter Search and Tabu Search methodologies. Computational comparisons are performed on a test-bed of 11 types of problems. For each type, four problems are considered, each one with dimension 50, 100, 200 and 500 respectively; thus totalling 44 instances. Our results show that the Scatter Tabu Search procedure is competitive with the state-of-the-art methods in terms of the average optimality gap achieved.

1. Introduction

From a naive standpoint, virtually all heuristics other than complete randomization induce a pattern whose present state depends on the sequence of past states, and therefore incorporate an implicit form of "memory." However, such an implicit memory, as indicated in [5], does not take a form normally viewed to be a memory structure. By contrast, the explicit use of memory structures constitutes the core of a large number of intelligent solving methods. They include tabu search, scatter search and evolutionary path relinking among others. These methods focus on exploiting a set of strategic memory designs. Tabu search (TS), the metaheuristic that launched this perspective, is the source of the term Adaptive Memory Programming (AMP) to describe methods that use advanced memory strategies (and hence learning, in a non-trivial sense) to guide a search. In linguistic terms, to define semantic hierarchies, we can say that AMP is the hyperonym of tabu search in a similar way that mathematical programming is the hyperonym of linear programming.

The unconstrained continuous global optimization problem may be formulated as follows:

$$(P) \quad \textit{Minimize} \quad f(x)$$
$$\textit{subject to} \quad l \le x \le u\,, \quad x \in \Re^n\,,$$

where *f(x)* is a nonlinear function and *x* is a vector of continuous and bounded variables. We investigated in [2] AMP methods for *P*; in particular we hybridized Scatter Search (SS) [7] and Tabu Search (TS) [5] for an efficient exploration of its solution space. In this paper we first summarize the main elements of this method, and then perform computational testing on the benchmark set of problems collected by Herrera and Lozano [6] for which global optima are known.

2. Scatter Search

The fundamental structure of our Scatter Search (SS) procedure is sketched in Figure 1. The method starts with the creation of an initial large set of diverse solutions *D* with the Diversification Generation Method.

Since we want the solutions in *D* to be diverse, we do not directly admit a generated solution *x* to become part of *D*, but only admit those with an Euclidean distance to the solutions already in *D, d(x,D),* larger than a pre-established distance threshold value *dthresh*. Therefore, in the initialization we generate solutions with a frequency mechanism until *DSize* solutions qualify to enter in *D*. Instead of the one-by-one selection of diverse solutions typically employed in Scatter Search to build the Reference Set, *RefSet*, we propose solving the max-min diversity problem (MMDP) [3] in the Step 5 of Figure 1.

Considering that the MMDP is a hard problem, we apply the *D*2 method [4] because it provides a good balance between solution quality and speed, attributes that are important in order to embed it as part of the overall solution method. The MMDP consists of finding, from a given set of elements (*D* in this case) and corresponding distances between elements, Euclidean in this case, the most diverse subset of a given size (the *RefSet* size *b*). The diversity of the chosen subset is given by the minimum of the distances between every pair of elements.

```
1.  Start with D = Pool = RefSet = Ø.
while (|D| < DSize) {
  2. Use the Diversification Generation Method
     to construct a solution x.
  3. If x ∉ D and d(x,D) > dthresh then add x
     to D (i.e., D = D ∪ {x} ).
  4. Evaluate the solutions in D and build
     RefSet = { $x^1$, …, $x^{b1}$ } with the best $b_1$
     solutions according to f.  NumEval = |D|
}
while (NumEval < MaxEval ) {
  5. Solve the MMDP in D to obtain $b_2$ diverse
     solutions ($b_2$=b-$b_1$) w.r.t. the solutions
     in RefSet.
  6. Build RefSet = { $x^1$, …, $x^b$ } with the $b_1$
     quality and $b_2$ diverse solutions.
  7. Evaluate the solutions in RefSet and
     order them according to their objective
     function value ($x^1$ is the best sol).
  8. Make NewSolutions = TRUE
     while ( NewSolutions ) {
       9.   Pool = Ø. NewSolutions = FALSE
       10.  Generate NewSubset (all pairs of
            solutions in RefSet that include at
            least one new solution.)
       while ( NewSubset ≠ ∅ ) {
           11.  Select the next subset s in
                NewSubset.
           12.  Apply the Solution Combination
                Method to s to obtain a new
                solution x.
           13.  Evaluate x. NumEval++
           14.  Add x to Pool.
           15.  Apply the Improvement Method to
                the best b solutions in Pool.
                Replace these b solutions with
                the outputs of the Improvement
                Method. Update NumEval adding
                the number of evaluations
                performed.
           16.  Delete s from NewSubset
       }
       while ( Pool ≠ ∅ ) {
           17.  Select the next solution x in
                Pool.
           18.  Let $y_x$ be the closest solution in
                RefSet to x.
           if  (f(x)<f($x^1$) or (f(x)<f($x^b$) &
                          d(x, $y_x$)>dthresh) )
                19.   Add x to the RefSet and
                      remove $x^b$.
                20.   Make NewSolutions = TRUE.
                21.   Remove x from Pool
       }
     }
     22. Remove the worst $b_2$ solutions from the
     RefSet
}
```

Figure 1. Outline of the SS procedure.

In the Step 4 of the method, the best b_1 solutions in terms of the objective function f are selected from D. Then, the b_2 (b_2=b-b_1) most diverse solutions in D found with the D2 method considering that b_1 solutions are already selected, are chosen in Step 5 to form the *RefSet*. The initialization of the SS algorithm finishes in the Step 6 with the construction of *RefSet* = $\{x^1,\ldots, x^b\}$.

The search is then initiated by assigning the value TRUE to the Boolean variable *NewSolutions*. In Step 10, *NewSubset* is constructed with all pairs of solutions in *RefSet*. The pairs in *NewSubset* are selected one at a time in lexicographical order and the Solution Combination Method is applied to generate a new solution in Step 12. The $(b_2$-$b)/2$ combined solutions are stored in a new set called *Pool*. The Improvement Method is applied to the best b solutions in *Pool* in Step 15. Each of these b solutions is replaced with the output of the Improvement Method.

If a solution x obtained by combination qualifies to enter *RefSet*, then, the worst solution x^b is removed from it (Step 19). The *NewSolutions* flag is switched to TRUE. If a new solution entered the *RefSet*, in the next main loop, when generating the pairs of solutions in the *RefSet* (Step 10), only pairs containing new solutions are included in *NewSubset*. Finally, when no new solutions are admitted to the reference set in the main while loop in Figure 1, the SS methodology dictates that the search either terminates or a *RefSet* rebuilding step is performed. The rebuilding step consists of eliminating all but the best b_2 reference solutions and reinitializing the process from the Step 5 in Figure 1. In our implementation, we have chosen to terminate the SS method after a pre-specified number of solution evaluations *MaxEval* as in most of the previous applications.

In the final steps of the SS algorithm, we test whether the solutions in *Pool* qualify to enter the *RefSet*. Given a solution x in *Pool*, let y_x be the closest solution to x in *RefSet*. We admit x to *RefSet* if it improves upon the best solution in it, x^1, or alternatively, if it improves upon the worst solution, x^b, and its distance to the closest solution in the *RefSet*, y_x, is larger than the pre-established distance threshold *dthresh* introduced above.

The Improvement Method is based on the so-called line-search. Given a solution x and an index i of a variable x_i, the line-search from x in the i-direction can be represented as the set $ls(x,h,i)$ where h is the width of the uniform grid of the discretized search space. This set contains all the feasible solutions that can be obtained in the h-grid by modifying variable x_i in solution x.

Given a solution x, we first order the variables in a random fashion and then perform the line search associated with each variable in this order. In other words, at a given step, we scan the set $ls(x,h,i)$ associated with variable x_i and if it contains a better solution than x, we move to this improved solution (and replace x with it). The improvement method performs iterations until no further improvement is possible

The Combination Method considers the line through two solutions x and y, given by the representation $z(\lambda) = x + \lambda (y - x)$, where λ is a scalar weight. We consider three points in this line: the convex combination z(1/2), and the exterior solutions z(-1/3) and z(4/3). We evaluate these three solutions and return the best as the result of the combination of x and y.

3. Tabu Search

Our AMP method couples the SS design described in Section 2 with Tabu Search elements. We have considered two alternative extensions for the improvement method; the first one introduces memory structures in the line-search scheme described above, and the second one replaces the line searches with an implementation of the Nelder-Mead simplex method [1], which we also modify by incorporating memory structures to improve its performance.

3.1 Tabu Line Search

Our improvement method consists of a short term Tabu Search procedure based on line-searches. A global iteration first orders the variables according to their attractiveness for movement in the current solution. Given a solution $x=(x_1, x_2,\ldots, x_i, \ldots, x_n)$, we compute for each variable x_i two associated solutions, $x^{i+h} =(x_1, x_2,\ldots, x_i+h, \ldots, x_n)$ and $x^{i-h} =(x_1, x_2,\ldots, x_i-h, \ldots, x_n)$. We then evaluate the attractiveness for movement of each variable x_i in x, $A(x,i)$, as

$$A(x,i) = \max \left(f(x)-f(x^{i+h}) , f(x)-f(x^{i-h}) \right).$$

Then, given an initial solution x, a global iteration of the improvement method first computes $A(x,i)$ for i=1 to n, and then orders the variables according to their A-values (where the variable with the largest A-value comes first). The first *ts* variables are selected one at a time in this order, and their corresponding line-searches are performed. Say for instance that j is the first variable in the list. Then we scan $ls(x,h,j)$ and select the best solution x' in this set. We then consider the second variable in the list, say x_k, and scan its associated line search from x': $ls(x',h,k)$ selecting the best solution x'' in this set if it improves upon the current solution, and so on.

As it is customary in Tabu Search implementations, we permit non-improving moves that deteriorate the objective value. Specifically, in the first step, the method selects the best solution x' in $ls(x,h,i)$ and the search moves from x to x', even if $f(x')>f(x)$. In a similar way, in the second step the method moves from x' to x'' thus defining the trajectory of the Tabu Search algorithm. Also note that when we select the second variable in the list, say k, to perform the move from x' to x'', its attractiveness, $A(x,k)$, is computed with respect to the initial solution x and we do not perform an update by computing $A(x',k)$, so that the attractiveness value $A(x,i)$ for any given variable x_i only represents an indicator. This is why the attractiveness information is updated after *ts* iterations and we do not explore additional variables in the list. At this point the search can either stop or continue with the next global iteration. In the latter case the A-values are first computed with respect to the solution obtained in the previous global iteration and the variables are ordered according to the values obtained.

Our Tabu Search algorithm implements a short term memory structure incorporating the following simple design. When a variable x_i is selected and we move to the best solution in its associated line-search, we label x_i tabu and we do not allow the method to select it during the next *tenure* iterations. Therefore, in each global iteration, the TS algorithm selects the first *ts* non-tabu variables in the list computed with the A-values. When TS finishes it returns as the output the best solution visited during its application. If no improvement has been found, it returns the initial solution as the output.

3.2 Nelder-Mead Simplex Search

The simplex search procedure of Nelder and Mead maintains a set of n+1 points, located at the vertices of a n-dimensional simplex. Each major iteration attempts to replace the worst point by a new and better one using reflection, expansion, and contraction steps. Given a solution x, our Improvement Method starts by perturbing each variable to create an initial simplex from which the local search begins.

Tabu restrictions in continuous spaces can be based both on direction and location, where location is usually expressed in terms of proximity to solutions previously visited. In our present work, we adopt a proximity criterion for generating tabu restrictions, which operates by reference to a distance threshold. Accompanying this, we use a simple memory structure to record all trial solutions we operate on with our modified simplex method. Then, at a given iteration, before applying the improvement method to a given trial solution, we test whether the trial solution lies within a hypersphere centered at any solution previously submitted to the simplex method (or centered at any of the perturbed solutions). If, so, the trial solution is considered tabu, and the improvement method is not applied. In order to reduce the computational effort associated with this memory structure, as in a customary short term tabu list, we limit the memory by maintaining a record only of the last *NumSol* solutions submitted to the simplex search.

4 Computational Experiments

The first set of experiments determines the key search parameters of our method. We refer the reader to the exhaustive preliminary experimentation in [2] to set the values of the search parameters as *h*=*MinRange*/100 where *MinRange* is the minimum of the variables ranges, (b_1 ,b_2)=(2, 6), and *dthresh*=*dgrid*/3 where *dgrid* is the distance grid. Here we only highlight one of these experiments in which we compare the following five alternative designs of our algorithm:

SS:	SS method described on Section 2,
SS+TS:	SS with Tabu Line Search,
SS+Sx:	SS with the original simplex method,
SS+TSx:	SS with TS simplex method,
STS:	SS+TS +TSx.

In this preliminary experiment we set the maximum number of solution evaluations *MaxEval* to 10,000 and we employ the following well-known 9 non-linear multimodal functions [7]: Branin, Rosenbrock(2), Shekel(5), Rastrigin(10), Rastrigin(20), Powell(24), Ackley(30), Beale and Powersum(8,18,44,114). As typically done in global optimization, we define the optimality gap as:

$$GAP = \left| f(x) - f(x^*) \right|$$

where x is a heuristic solution and x^* is the optimal solution. We have implemented our procedures in C and all the experiments were conducted on a Pentium 4 computer at 3 GHz with 3 GB of RAM.

Method	Avg. GAP	# Optima
SS	0.029	7
SS+TS	0.004	7
SS+Sx	0.001	8
SS+TSx	0.001	8
STS	0.000	9

Table 1. Scatter Search variants

Table 1 shows that the five variants considered are able to provide high quality results for this problem, since the average gap values are, in all the cases, below 0.1%. Moreover, comparing SS with SS+TS we can see that significant marginal improvement is achieved by replacing the simpler form of the line-search with the tabu line-search in the Scatter Search algorithm. Further, comparing SS+Sx with SS+TSx, we see the advantage of including a memory structure to modify the improvement method (the Nelder-Mead simplex method in this case), so that successive applications of the method are restricted to operate only with solutions relatively far from those already submitted to the improvement method. Finally, the STS method, which couples the SS+TS with the TS-modified Nelder-Mead simplex procedure outperforms the other Adaptive Memory Programming variants. Thus, in sum, the combination of two different improvement methods provides the best results, producing a variant that is able to obtain optimal solutions to all of the 9 instances tested.

Having determined the values of the key search parameters for our algorithm in the first set of experiments, we perform a second set of experiments to execute the best variant, STS, over the proposed set of 11 scalable function optimization problems by Herrera and Lozano [6]:

- F1 to F6 functions of the CEC'2008 [8] test suite.
- Schwefel's Problem 2.22 (F7), Schwefel's Problem 1.2 (F8), Extended f10 (F9), Bohachevsky (F10), and Schaffer (F11).

Following the guidelines of the organizers, STS is run for 25 independent times on each instance.

	Min	Max	Mean
F1	0.13	0.64	0.25
F2	74.87	94.23	86.08
F3	102.38	384.62	208.23
F4	0.16	0.49	0.26
F5	1.09	1.13	1.12
F6	0.08	0.25	0.14
F7	0.00	0.00	0.00
F8	587.69	1915.55	1159.37
F9	0.00	0.40	0.03
F10	0.00	0.00	0.00
F11	0.00	0.35	0.05

Table 2. Results with n=50

The study is performed with dimensions n=50, 100, 200 and 500 (i.e., the 11 functions above are instantiated for each value of n considered), and the maximum number of evaluations is 5,000n. Each run stops when the maximum number of evaluations is achieved. We then record the best value (Min), the worst value (Max) and the mean value (Mean) over the

25 runs for each instance. Tables 2 to 5 report the results for each dimension respectively.

	Min	Max	Mean
F1	0.84	4.48	2.12
F2	87.67	118.51	106.19
F3	698.94	1156.16	897.29
F4	1.75	4.23	2.99
F5	1.23	1.26	1.24
F6	1.22	1.48	1.38
F7	0.00	0.00	0.00
F8	4253.12	13217.69	7581.45
F9	0.00	0.46	0.09
F10	0.00	0.00	0.00
F11	0.00	0.61	0.21

Table 3. Results with *n*=100

	Min	Max	Mean
F1	28.30	52.49	37.04
F2	104.58	134.19	123.08
F3	2391.61	5131.75	3340.09
F4	19.60	26.67	22.67
F5	1.48	1.55	1.53
F6	1.09	1.71	1.60
F7	0.00	0.00	0.00
F8	23070.83	52254.88	33918.05
F9	0.09	3.78	0.61
F10	0.00	0.00	0.00
F11	0.00	15.50	2.09

Table 4. Results with *n*=200

	Min	Max	Mean
F1	146.42	152.99	150.40
F2	133.05	151.20	144.05
F3	7833.68	14696.60	11409.55
F4	66.79	83.85	80.46
F5	2.30	2.42	2.37
F6	1.25	1.83	1.71
F7	0.00	0.00	0.00
F8	153165.44	325991.44	241959.97
F9	1.18	47.38	16.96
F10	0.00	0.00	0.00
F11	6.21	67.63	27.89

Table 5. Results with *n*=500

As expected, the results in these tables confirm that the larger the dimension the more difficult the instance. The STS method provides optimal or near optimal solutions for problems F1, F4 to F7 and F9 to F11 (especially in lower dimensions). Problems F2, F3 and F8 emerged as very difficult to solve, especially F8 for which our method seems inadequate.

5. Conclusions

We have described the development and implementation of an adaptive memory programming procedure integrating Scatter Search and Tabu Search for unconstrained nonlinear optimization. Our experimentation shows that we significantly improve the local search methods by introducing a memory structure. This is especially true for the line search based method, but we were also able to appreciably improve the popular Nelder and Mead simplex method. Moreover, our study reveals that a combination of line search with the simplex algorithm, when both are equipped with a suitable memory structure, produces high quality outcomes for the preponderance of the test functions considered.

Acknowledgments

This research has been partially supported by the *Ministerio de Ciencia e Innovación* of Spain (TIN2006-02696, TIN2009-07516) and by the Comunidad de Madrid (URJC/TIC-3731). The authors would like to thank Profs. Herrera and Lozano for their invitation to perform this study.

References

[1] Avriel, M. 1976. *Nonlinear Programming, Analysis and Methods*, Prentice-Hall, Englewood Cliffs, New Jersey.

[2] Duarte, A., Martí, R. and Glover, F., 2009. Hybrid Scatter Tabu Search for Unconstrained Global Optimization, *Annals of Operations Research*, to appear.

[3] Resende, M., R. Martí, M. Gallego and A. Duarte, 2009. GRASP and Path Relinking for the Max-Min Diversity Problem, *Computers and Operations Research*, to appear.

[4] Glover, F., C.C., Kuo and K.S. Dhir, 1998. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences* 19 (1), 109-132.

[5] Glover, F. and M Laguna, 1997 Tabu Search, Kluwer Academic Publishers, Boston.

[6] Herrera, F. And M. Lozano, 2009, Workshop on Evolutionary Algorithms and other Metaheuristics for Continuous Optimization Problems - A Scalability Test, http://sci2s.ugr.es/programacion/workshop/Scalability.html

[7] Laguna, M., R. Martí, 2003. *Scatter Search – Methodology and Implementations in C.* Kluwer, Boston, MA.

[8] Tang, K., X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark for the CEC'2008. Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007