Adaptive Memory Programming for Constrained Global Optimization

LEON LASDON — Information, Risk, and Operations Management Department
The University of Texas at Austin, USA
Leon.Lasdon@mccombs.utexas.edu

ABRAHAM DUARTE — Departamento de Ciencias de la Computación
Universidad Rey Juan Carlos, Spain
Abraham.Duarte@urjc.es

FRED GLOVER — OptTek Systems, Inc.
Boulder, CO 80302, USA
Glover@opttek.com

MANUEL LAGUNA — Leeds School of Business
University of Colorado at Boulder
Laguna@colorado.edu

RAFAEL MARTÍ — Departamento de Estadística e Investigación Operativa
Universidad de Valencia, Spain
Rafael.Marti@uv.es

Abstract

The problem of finding a global optimum of a constrained multimodal function has been the subject of intensive study in recent years. Several effective global optimization algorithms for constrained problems have been developed; among them, the multistart procedures discussed in Ugray et al. (2007) are the most effective. We present some new multistart methods based on the framework of adaptive memory programming (AMP), which involve memory structures that are superimposed on a local optimizer. Computational comparisons involving widely used gradient-based local solvers, such as Conopt and OQNLP, are performed on a testbed of 26 problems that have been used to calibrate the performance of such methods. Our tests indicate that the new AMP procedures are competitive with the best performing existing ones.

Last revision: November 21, 2008

1. Introduction

The constrained continuous global optimization problem (*P*) may be formulated in a general form as follows:

$$
\begin{array}{lll}
(P) & \text{minimize} & f(x) \\
 & \text{subject to:} & \\
 & & G(x) \leq b \\
 & & x \in S \subset R^n
\end{array}
$$

where x is an n-dimensional vector of continuous decision variables, G is an m-dimensional vector of constraint functions, and without losing generality the vector b contains upper bounds for these functions. The set S is defined by simple bounds on x, and we assume that it is closed and bounded, i.e., that each component of x has a finite upper and lower bound. The objective function f and the constraint functions G are assumed to have continuous first partial derivatives at all points in S.

There are many effective methods, such as Conopt, Snopt and Knitro, most of them requiring gradients of the problem functions, for finding local solutions to (*P*). In this paper we propose memory structures (Glover 1989) that can be superimposed to these methods to drive the search on a long term horizon to solve the global optimization problem. These structures use a local solver to generate trial solutions which are candidates for a global optimum, where as customary the best feasible candidate is retained as the overall "winner". The local solver may be gradient-based, or may be a "black box" algorithm as typically used in simulation optimization (see, e.g., April et al., 2006; Better, Glover and Laguna, 2007). The local solver used here is Conopt, a GRG algorithm (Smith and Lasdon, 1992), described in (Drud 1994). This gradient-based algorithm is known to have a superlinear convergence rate, and has an excellent combination of reliability and efficiency, when applied to both small and large sparse smooth NLP's. Our adaptive memory structure will guide the selection of starting points for the local solver. Specifically, we propose two different Adaptive Memory Programming (AMP) approaches (Glover and Laguna, 1997); the first one, described in Section 2, is based on a *tabu tunneling strategy* and the second one, described in Section 3, is based on a *pseudo-cut strategy*. Both are designed to prevent the search from being trapped in local optima.

In Section 4, we describe the GAMS implementation of both AMP approaches. Section 5 is devoted to our computational experiments on 26 hard problems from the literature (see Ugray et al. 2007). We will show that the *tabu tunneling strategy* is able to outperform the state-of-the-art methods for constrained global optimization. On the other hand, the *pseudo-cut strategy* obtains lower quality results on average (as compared to the tunneling strategy), although it substantially improves the Conopt method and it is capable of establishing new best-known solutions to three instances. We also show that the pseudo-cut strategy can be coupled with the tunneling method for improved outcomes. The paper finishes with relevant conclusions.

2. Tabu Tunneling Method

Metaheuristic methodologies diversify a search to march beyond local optimality in the quest for a global optimum. Applying a local optimizer to points in the same region of the solution space may result in repeatedly obtaining the same local optimum, unless additional constraints are imposed or the evaluation criterion is modified. In this section we propose an adaptive modification of the objective function to perform multiple searches with the goal of reaching different local optima of the original objective function.

Our approach modifies and extends the global optimization tunneling method (TM) of Levy and Gomez (1985) for unconstrained problems. The original TM approach consists of two phases, minimization and tunneling, which are alternated until a cutoff point is reached. For a given starting point s_0, the minimization phase applies a descent algorithm to find a local optima s. The tunneling phase is designed to obtain a good starting point for the next minimization phase. The objective is to obtain a point t with a value as good as s ($f(t) \leq f(s)$) in a different basin of attraction. Therefore if we apply the descent method from t we can obtain now a solution w better than s and t as shown in Figure 1.

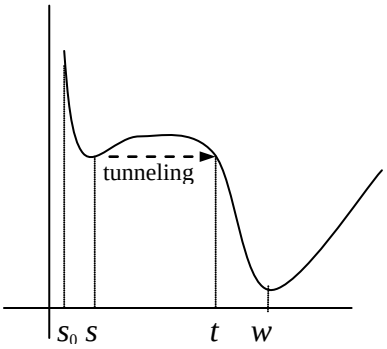

Figure 1. Tunneling approach

The objective of the tunneling problem must therefore be constructed so that its minimization leads to points which both improve the original function $f(x)$ and are distant from the current solution s. A standard tunneling function $T(x,s,\lambda)$ is given by the expression:

$$T(x,s,\lambda) = \frac{f(x) - f(s)}{\|x - s\|^{\lambda}}$$

where λ is positive. Levy and Gomez proposed that an equation-solving method be applied to seek a point where $T = 0$.

As mentioned in Murray and Ng (2002), a pure tunneling method has severe drawbacks, because the T function inherits any nonconvexity in f, and the new term in the denominator creates a singularity at s. Thus the tunneling subproblem is at least as

difficult as the original, and an equation solver or local optimizer applied to it may fail to find a point t satisfying $f(t) \le f(s)$ even if one exists.

To overcome these limitations, and to extend the method to constrained problems, our tabu tunneling method (TTM) modifies the original tunneling procedure as follows:

1. Since it is difficult to find a point t different from a local solution s satisfying $f(t) \le f(s)$ in a single application of a local solver, we have developed a multistep process for this purpose, which we call the tunneling loop. We define a *TabuList* of points from which to move away, including both the most recent local solution of the original problem *P*, and recent solutions of tunneling subproblems which failed to achieve the desired condition.

2. To enforce movement away from several points, we redefine the denominator of T to be the product of distances from the points in *TabuList.*

3. Rather than making the current best objective value *Best_f* the target to be achieved by the tunneling loop, we define an *aspiration* value for the objective function which is slightly less than *Best_f*;

$$aspiration = Best\_f - \varepsilon_1 \cdot (1+\text{abs}(Best\_f))$$

   where the default value of ε_1 is 0.02, and the additive unity term is included to ensure a nonzero reduction when *Best_f* is zero, and a significant reduction when abs(*Best_f*) is very small. With these elements, our tunneling function *TTf* is:

$$TTf(x) = \frac{(f(x) - aspiration)^2}{\prod_{s_i \in TabuList} (dist(x, s_i))^2}$$

   For simplicity, we do not indicate explicitly the dependence of *TTf* on *aspiration* and the points in *TabuList.* This function has a global minimum of zero at any feasible point where *f* = *aspiration,* and minimizing this expression tends to move away from all points in *TabuList*.

4. To accommodate constraints other than simple bounds, we redefine the tunneling subproblem (*TP*) to include all the constraints of the original problem:

$$\begin{aligned} (TP) \quad & \text{minimize} \quad TTf(x) \\ & \text{subject to:} \\ & \qquad G(x) \le b \\ & \qquad x \in S \end{aligned}$$

5. Since *TTf* is not defined at any of the points in *TabuList*, the starting point for (*TP*) cannot be in *TabuList.* Further, if the local solver used to solve (*TP*) encounters any point in *TabuList* during its solution process, it may terminate due to a domain

violation error. This happened when using Conopt in a number of experiments, for the following reasons. Since Conopt is a "feasible point" solver, if the starting point is not feasible, it tries to find a feasible point using a standard phase 1 procedure, just as the Simplex algorithm for linear programming would do. This phase 1 procedure ignores the given objective and substitutes an objective equal to the sum of constraint violations. Thus there is no mechanism in phase 1 to avoid the points in *TabuList*, and we found that in many cases where an infeasible starting point was supplied for (*TP*), Conopt's phase 1 terminated with a feasible solution equal to a point in *TabuList.* When it attempted to use this point to initiate minimization of *TTf* in phase 2, this triggered a fatal error message and Conopt terminated.

The only sure way to avoid the problem in point 5 above is to provide the local solver with a feasible starting point not equal to any of the points in *TabuList.* We have found that an effective way to do this is to first randomly perturb away from the current local solution (the newest point in *TabuList*) to a (generally infeasible) point x_0, and then find the feasible point closest to x_0 by solving the following *Projection Problem*

$$\begin{aligned} (PP) \quad & \text{minimize} \quad \|x - x_0\|^2 \\ & \text{subject to:} \\ & \qquad G(x) \le b \\ & \qquad x \in S \end{aligned}$$

In all constrained problems solved thus far, containing hundreds of instances of (*PP*), this strategy has never failed to provide a suitable starting point for (*TP*).

As in a variety of simple tabu search implementations, we maintain *TabuList* as a circular list. After each solution of the tunneling problem defined above, we add the solution to that problem to the list, and if the size of the new list exceeds the limit (called *TabuTenure* in the following description), some old point is dropped from the list. We have experimented with dropping the oldest element, dropping the element farthest from point just added to the list, and dropping the oldest element whose distance from the point just added is larger than a specified fraction of the largest distance. Results comparing these alternatives are discussed later.

Figure 2 provides a pseudo-code description of our tabu tunneling algorithm, in which LS is the local search or descent algorithm for constrained global optimization. In line 5 we use the expression s = LS(s_0, $f(x)$,G,S) to indicate that we optimize problem (*P*) (i.e., the minimization of $f(x)$ subject to the original constraints G and S) from the initial point s_0 with the LS optimizer. Similarly, in line 11 the expression s = LS(s', *TTf*(*x*,*aspiration*),G,S) indicates that we solve problem (*TP*) with objective function *TTf*(*x*) subject to the original constraints G and S from the initial point s' using LS. Also, s = LS(x_0, $\|x - x_0\|^2$,G,S) in line 12 represents solving the projection problem (*PP*) starting from x_0 using LS.

The *While*-loop (lines 9 to 18) in Figure 2 performs the tunneling approach, and is repeated *TotalIter* times. Note that in line 15 we update the *TabuList*, which means that we add the point *s* obtained in line 14 and drop some other point from the list. The tunneling loop ends when the tunneling solution has an *f* value less than or equal to *Best_f*, or when the iteration limit *MaxIter* is reached. At this point the search can either terminate (if the *GlobalIter* counter reaches its maximum, *TotalIter*) or continue performing a new iteration in Step 5 from s_0 (the latest local optimum obtained with the tunneling minimization).

Initialization

1. s_0 = user-provided initial solution
2. Let *TabuList* be the memory list with *TabuTenure* size
3. Let *Best_f* be the value of the best solution found (initialized to ∞)
4. *GlobalIter* = 0

Minimization of original objective starting from s_0

5. s = LS(s_0, $f(x)$,G,S)
6. *TabuList* = {s}

If ($f(s) < Best\_f$)

7. $Best\_f = f(s)$

Tabu Tunneling

8. $i = 0$, *improve* = 0

While ($i < MaxIter$ **and** *improve* = 0) // default value of *MaxIter* = 5

9. r = vector with components uniformly distributed on [-1,11]
10. $\beta = \varepsilon_2 * \|s\| / \|r\|$ // default value of ε_2 = 0.1
11. $x_0 = s + \beta * r$
12. s' = LS(x_0, $\|x - x_0\|^2$,G,S)
13. *aspiration* = $Best\_f - \varepsilon_1 \cdot (1+\text{abs}(Best\_f))$ // default value of ε_1=0.02
14. s = LS(s', *TTf*(x,*aspiration*),G,S)
15. Update the *TabuList*

If ($f(s) \leq Best\_f$)

16. $Best\_f = f(s)$
17. *improve*=1

18. $i = i + 1$
19. *GlobalIter* = *GlobalIter* + 1

If (*GlobalIter* = *TotalIter*) // default value of *TotalIter* = 20

20. STOP

Else

21. $s_0 = s$ and GOTO Step 5.

Figure 2. Pseudo-code of the Tabu Tunneling method

3. The Tabu Cutting Method

The pseudo-cut strategy (Glover 2008) is based on generating hyperplanes that are orthogonal to selected rays (half-lines) originating at a point x' and passing through a second point x'', so that the hyperplane intersects the ray at a selected point x^o. We

employ a simplified variant of the method here. The half-space that forms the pseudo-cut is then produced by the associated inequality that excludes x' and x'' from the admissible half-space. Let x identify points on the ray originating at x' that passes through x'':

$$x = x' + \lambda(x'' - x'), \qquad \lambda \geq 0$$

A hyperplane orthogonal to this line may then be expressed as $(x'' - x')\, x = b$ where b is an arbitrary constant. Let x^o be a point in the ray ($x^o = x' + \lambda^o(x'' - x')$) then, the inequality (pseudo-cut) that excludes x' and x'' and includes x^o is given by:

$$(x'' - x')\, x \geq (x'' - x')\, x^o$$

We make use of this inequality or pseudo-cut (shown in Figure 3) called $pcut(x')$ within a two-stage process. In the first stage x' represents a point that is used to initiate a current search by the local search or descent method LS, and x'' is the point obtained. We can use GRG methods or any other local optimizer for constrained optimization as the descent method and employ the same notation introducing in the previous section (i.e. $x'' = \mathrm{LS}(x', f(x), G, S)$.

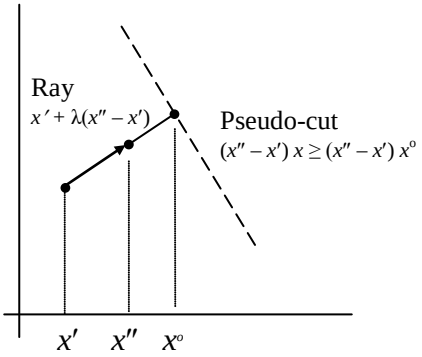

Figure 3. Pseudo-cut representation

Once we obtain the point x'' with the local optimizer, we generate a new point x^o. The point x^o can be simply computed as $x^o = x' + m(x'' - x')$ where m is a multiple of the distance between x' and x''. We now add the pseudo-cut $pcut(x')$ to the set of constraints of the problem and solve the extended problem by applying the local search method from x^o to obtain y. Using our previous notation:

$$y = \mathrm{LS}(x^o, f(x), G \cup \{pcut(x')\}, S).$$

We repeat this first stage of the process iteratively. We make now $x' = x^o$, $x'' = y$, compute a new point x^o and create the associated pseudo-cut. Since we want to solve now the original problem with two pseudo-cuts (the one introduced in the previous iteration and the new one), we create a short term memory structure *TabuCutList* to store the recent cuts (we limit its size to the *TabuTenure* latest cuts). As it is customary in tabu search implementations the short term memory is implemented as a circular list in which

we drop from the list the first cut (the oldest one) when we introduce a new one, thus keeping the size *TabuTenure* constant.

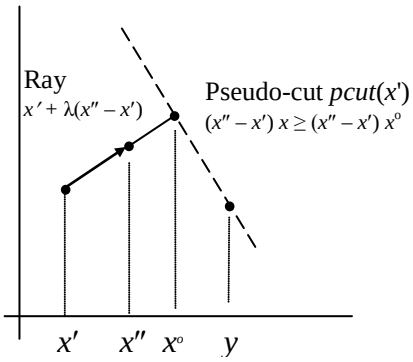

Figure 4. Second stage of the method

In the process above, we can eventually obtain the point $y = \text{LS}(x^o, f(x), G \cup \{pcut(x')\}, S)$ lying on the hyperplane associated with the current pseudo-cut $pcut(x')$, as shown in Figure 4. In this case, we resort to the second stage of our algorithm, in which we replace $pcut(x')$ with a new cut. To this end we define a new point x^o in the segment from x'' to y: $x^o = x'' + m(y - x'')$. Note that the multiple m is selected to be greater than 1 in the first stage and less than 1 in this second stage. Then the new cut is given by the expression:

$$(y - x'')\, x \geq (y - x'')\, x^o$$

As in the first stage, we solve now the problem with this cut (that we called $pcut(x'')$). In terms of our notation, we apply $\text{LS}(x^o, f(x), G \cup \{pcut(x'')\}, S)$. If the final point of this local search application does not lie on the hyperplane associated with the current cut, we return to the Stage 1 of the process; otherwise we perform a new iteration of the Stage 2.

At a given iteration of the *tabu cutting method* (TCM) we apply the local search method LS from x', to solve the original problem in which we added all the cuts in the *TabuCutList*. That is, we obtain y by applying the local optimizer with the following parameters:

$$y = \text{LS}(x', f(x), G \cup TabuCutList, S)$$

In a basic design of the TCM we repeat this mechanism, in which we add one cut at each iteration (dropping from the list of active cuts the oldest one) and apply the local search method for *MaxIter* iterations. However, we have empirically found that this basic mechanism is not able to improve the original solutions by itself, and a more elaborated managing of the cuts is needed.

Figure 5 shows a pseudo-code of TCM. Given that the points x' and x'' change their identities in the two stages, it is convenient to refer to the points as $P0$ ($P0 = x'$ in the first stage), $P1$ ($P1 = x''$ in the first stage), $Q1$ ($Q1 = x^0$ in the first stage), etc.

Initialization

1. Let *TabuCutList* be the memory list of pseudo-cuts with *TabuTenure* size
2. Let m=0.1, *MaxIter* = 5 and *GlobalIter*= 20
3. Let $P0$ be a *random point and Best_f = f(P0)*

While (*Iter*1 < *GlobalIter*)

4. *TabuCutList* = ∅
5. *improved* = FALSE
6. *pert* = m
7. $P1$ = LS($P0$, *f(x),G,S*)

If (*f*($P1$) < *Best_f*)

8. *Best_f* = *f*($P1$)

9. $Q1 = P0 + (1 + pert)*(P1 - P0)$

While (*Iter*2 < *MaxIter* **and** NOT *improved* **and** ||P1 – P0|| > *minDist*)

10. Remove from *TabuCutList* the cuts violated at $Q1$ and the old cuts (*TabuTenure*)
11. Add to *TabuCutList* the cut *pcut*($Q1$): $(P1 - P0)\, x \geq (P1 - P0)\, Q1$
12. $Q2$ = LS($Q1$, *f(x),G*∪*TabuCutList, S*)

If (*f*($Q2$) < *Best_f)*

13. *improved* = TRUE
14. *Best_f* = *f*($Q2$)
15. go to 21

If (||*Q1 – Q2*|| < *minDist*)

16. *pert* = *pert* + m

Else

17. *pert* = m

// Stage 1

If ($Q2$ does not verify *pcut*($Q1$) with equality **or** $\|Q2 - Q1\|$ < *minDist*)

18. $P0 = Q1$; $P1 = Q2$; $Q1 = P0 + (1 + pert)*(P1 - P0)$

//Stage 2

If ($Q2$ verifies *pcut*($Q1$) with equality **and** $\|Q2 - Q1\|$ > *minDist*)

19. $P0 = P1$; $P1 = Q2$; $Q1 = P0 + (1 - pert)*(P1 - P0)$

20. *Iter*2 = *Iter*2 + 1

21. Generate a new point $P0$ perturbing $Q2$

Figure 5. Pseudocode of the pseudo-cut method

Steps 10 to 20 in Figure 5 show the main loop of our Tabu Cutting Method. We can identify the following three cases when applying the local search method LS from $Q1$ to obtain $Q2$. Let *pcut*($Q1$) be the latest cut added to *TabuCutList* (Step 11 in the procedure). Depending where $Q2$ lies we can distinguish:

A. $Q2$ is equal to $Q1$. This case is identified in the IF statement above step 16 (||*Q1 – Q2*|| < *minDist*). This indicates that the local search method is not able to generate a new solution, probably because the search remains in the same basin of attraction. We therefore add a new cut farther away from x''. Then, in Step 16 we increase the value of *pert*, and resort to Step 18 to compute a new point $Q1$ as shown in Figure 6.

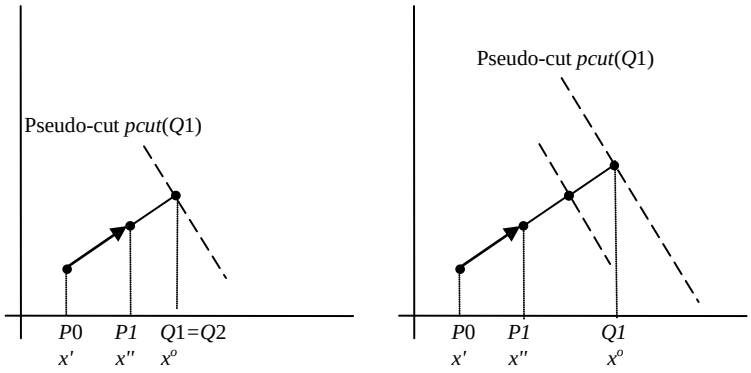

Figure 6. A-Strategy

B. $Q2$ verifies the cut $pcut(Q1)$ with equality $((P1 - P0)\ Q2 \geq (P1 - P0)\ Q1)$ but it is different to $Q1$ as shown in Figure 7. This indicates that we probably would find better solutions closer to $P1$ (in the infeasible region of the added cut). This is the Stage 2 of the method. In Step 19 we define a new point $Q1$ closer to $P1$ (according to *pert* = *m* set in Step 17) and in Step 10 we remove $pcut(Q1)$.

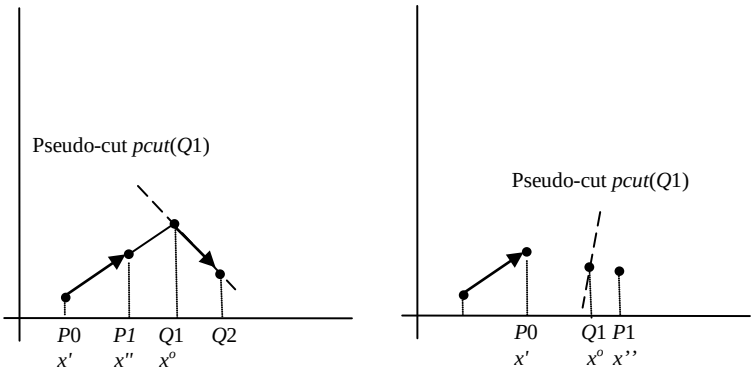

Figure 7. B-Strategy

C. $Q2$ strictly verifies the cut with inequality $((P1 - P0)\ Q2 > (P1 - P0)\ Q1)$. We therefore have obtained a good solution ($Q2$) in the feasible region of the cut $pcut(Q1)$. So we keep this cut in the *TabuCutList* and repeat the process from $Q2$. Then, in Step 18 we exchange the roles of the points ($P0 = Q1$; $P1 = Q2$), and compute a new point $Q1$. Figure 8 shows this case.

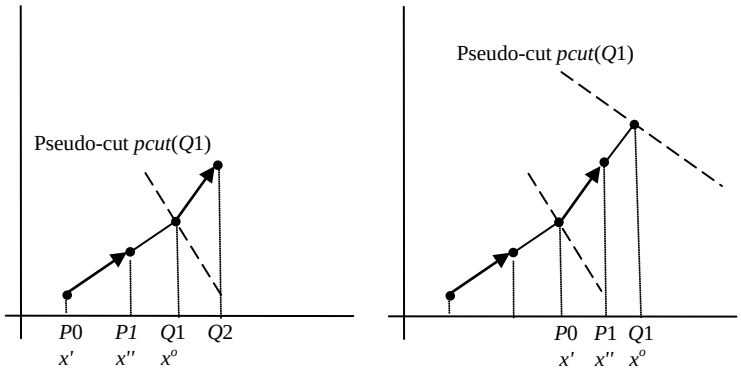

Figure 8. C-Strategy

The inner *While* loop terminates when the maximum number of iterations *MaxIter* is reached, or the value of the incumbent solution (*Best_f*) is improved. Then, a new point *P*0 is generated in Step 21 by perturbing *Q*2. Specifically if l_i and u_i are respectively the lower and upper bounds of variable x_i, and r is a random number in the range [0,1], then the value of the i^{th} variable in *P*0 is computed from *Q*2 as:

$$P0_i = Q2_i + r\,(u_i - l_i)$$

Then the search continues in the outer *While* loop until *GlobalIter* global iterations are performed.

4. GAMS Implementation

We implemented both AMP approaches — TTM and TCM — in the GAMS algebraic modeling language (see www.gams.com). While GAMS is usually employed to code an optimization model, it contains enough programming constructs, e.g. loops and conditionals, to quickly and easily implement algorithms of moderate complexity. Alp, Ertek and Birbil (2006) describe GAMS implementations of column generation for cutting stock problems and Conejo et al. (2006) develop GAMS code for decomposition approaches including Dantzig-Wolfe and Benders. GAMS's advantages in this respect include:

1. Applying a GAMS solver to a GAMS model requires a single SOLVE statement, and one additional statement allows one to change the solver. This permitted us to try any of 3 different local NLP solvers within the AMP framework. In a more general programming language like Java or C, each solver generally has differing input and interface requirements, so employing multiple solvers is much more difficult.
2. Because of advantage 1, as well as the powerful and compact way one represents optimization models in GAMS, someone with a moderate knowledge of GAMS can implement and debug an algorithm such as the three referred to in the references above or AMP in a few days.
3. The algorithm can be applied to any problem already coded in GAMS. Since GAMS has a library currently containing 413 smooth NLP's (called GLOBALLIB), most

with multiple distinct local optima, solution methods can be applied to a fairly large set of problems, many of which are large and/or difficult. GLOBALLIB can be downloaded at http://www.gamsworld.org/global/globallib.htm

4. GAMS is already interfaced to four powerful and widely used global optimizers, OQNLP, BARON, LGO, and LINDOGLOBAL, so these can also be applied to any problem solvable by the AMP GAMS implementation, and the results are directly comparable.
5. AMP can be started from the final solution of any other GAMS solver, showing what happens when the two are combined sequentially. The experiments below include a set starting with OQNLP, then applying TTM, as well as TTM coupled with TCM.

A significant disadvantage of the GAMS implementation of TTM arises because the objective of problem (*TP*), the function *TTf*(*x*), depends on *x* not only through the original objective *f*(*x*) but also directly because of its denominator. Thus models with different names for the variables *x* require different GAMS expressions to compute the denominator of *TTf*. Hence the algorithm implementation depends on the model, restricting the number of problems to which this implementation can be easily applied. We chose two families of problems, a set of bound-constrained "atom energy" problems where the GAMS variable names are *x*(*i*), *y*(*i*), and *z*(*i*), and a more diverse set of more generally constrained problems, where the variable names are *x*(*i*) (with the exception of EX2_1_8 where the names are *x*(*i*,*j*)).

5. Computational Experiments

This section describes the results obtained applying the GAMS implementation of TTM and TCM described above. All experiments were conducted on a Dell Latitude D820 computer with an Intel T2500 CPU running at 2.0 GHz with 1.9 GB of RAM. The GAMS version used was 22.5.

We have employed 26 problem instances in our experimentation. Results obtained solving these using the OQNLP multistart algorithm are given in Ugray et al. (2007). The problems are members of both the GAMS GLOBALLIB and the set compiled by Floudas et al. (1999). Table 1 shows the characteristics of these problems, providing the name, number of variables, number of linear and nonlinear constraints, best known objective value, and the value obtained when running the Conopt solver from the starting point provided in the GAMS model (Conopt Value). The "atom energy" problems referred to above choose the locations of a cluster of *n* particles to minimize the potential energy of the cluster, using two different potential energy functions, called Lennard-Jones (EX8_6_1_n) and Morse (EX8_6_2_n). The decision variables are the (*x*,*y*,*z*) coordinates of each particle. Particle 1 is located at the origin, and three position components of particles 2 and 3 are fixed, so each family of problems has 3*n* variables, with 6 of them fixed. These problems have many local minima, and their number increases rapidly with problem size, *n*, so they constitute a good test for global optimization algorithms. Eight problems from each set are solved, with *n* ranging from 5 to 50. The Lennard-Jones series has many nonlinear constraints, imposed to insure that a division by zero in the objective (which occurs when two or more particles occupy the

same point) is avoided, and all have simple bounds. None of these constraints are active at any local solution, so both series are "essentially unconstrained". The remaining 10 problems have either general linear constraints or linear and nonlinear constraints. Some of these were selected because OQNLP had difficulties solving some of them in the experiments in (Ugray et al. 2007).

No.	Problem	Vars.	Linear constraints	Nonlinear constraints	Best known objective	Conopt value	Conopt gap
1	EX2_1_1	5	1	0	-17	0	100.00%
3	EX2_1_6	10	5	0	-39.00	-21.25	45.83%
3	EX2_1_8	24	10	0	15639.00	19971	90.24%
4	EX2_1_9	10	1	0	-0.375	-.333.00	15.26%
5	EX2_1_7_1	20	10	0	-394.75	-38.52	30.85%
6	EX2_1_7_2	20	10	0	-884.75	-794.77	16.07%
7	EX2_1_7_3	20	10	0	-8695.01	-6012.66	90.18%
8	EX2_1_7_4	20	10	0	-754.75	-633.44	27.70%
9	EX2_1_7_5	20	10	0	-4150.41	-407.43	11.20%
10	EX3_1_3	6	4	2	-310.00	-132	57.42%
11	EX8_6_1_5_	15	0	10	-9.10	-9.10	0.00%
12	EX8_6_1_10	30	0	45	-28.42	-26.02	10.85%
13	EX8_6_1_15	45	0	105	-52.32	-47.45	9.31%
14	EX8_6_1_20	60	0	190	-77.18	-70.67	8.43%
15	EX8_6_1_25	75	0	300	-102.37	-95.47	6.74%
16	EX8_6_1_30	90	0	435	-128.29	-113.75	11.33%
17	EX8_6_1_40	120	0	780	-184.16	-168.34	8.59%
18	EX8_6_1_50	150	0	1225	-241.67	-232.79	3.67%
19	EX8_6_2_5_	15	0	0	-9.30	-9.30	0.00%
20	EX8_6_2_10	30	0	0	-31.89	-31.07	2.56%
21	EX8_6_2_15	45	0	0	-63.16	-63.16	0.00%
22	EX8_6_2_20	60	0	0	-97.42	-97.42	0.00%
23	EX8_6_2_25	75	0	0	-136.07	-135.03	0.77%
24	EX8_6_2_30	90	0	0	-177.58	-177.46	0.07%
25	EX8_6_2_40	120	0	0	-268.39	-267.74	0.24%
26	EX8_6_2_50	150	0	0	-366.64	-366.64	0.00%
	Total						547.32%

Table 1. Problem characteristics

The last column in Table 1 shows the Conopt "gap", the percentage difference between the best known objective value and the final Conopt objective value. Conopt alone is able to achieve the best known objective value in only 6 of these problems, and 16 of Conopt's solutions have a gap greater than 5%. The sum of the gaps is 547.3%.

5.1. Base Case Runs

Table 2 shows the results obtained with the TTM and TCM methods. Both are run using iteration limits of *TotalIter* = 20 major iterations, *MaxIter* = 4 minor iterations per major iteration and a *TabuTenure* value of 2. Thus there are 20 tunneling loops in the TTM method, each ending with a solution of the original problem, starting from the final point

reached in the tunneling loop. In each tunneling loop the projection and tunneling subproblems are solved at most 4 times, fewer if the best value found so far is achieved or exceeded before 4 tunneling subproblems are solved. Each such occurrence is a tunneling "success". Hence the local solver Conopt is called upon to find 21 solutions of the original problem (one at the user-provided initial point) and to solve at most 4*20 = 80 subproblems and 80 projection subproblems, at most 181 solver calls in all. Similarly, there are 20 cutting loops in the TCM, each one with the addition of at most 4 pseudo-cuts.

	TTM					TCM				
Problem number	**Best value**	**Gap**	**Iter to best**	**Solver calls**	**Tunnel successes**	**Best value**	**Gap**	**Iter to best**	**Solver calls**	**Cut successes**
1	-17.00	0.00%	20	129	11	-13.00	23.53%	18	60	2
2	-39.00	0.00%	8	53	6	-39.00	0.00%	1	19	0
3	15639.00	0.00%	2	7	4	15639.00	0.00%	20	20	2
4	-0.375	0.00%	1	4	8	**-0.403**	**-7.47%**	2	20	1
5	-394.75	0.00%	15	100	7	**-477.33**	**-20.92%**	40	87	3
6	-884.75	0.00%	2	15	4	-832.84	5.87%	2	20	1
7	-8695.01	0.00%	6	45	5	-7275.38	16.33%	2	20	1
8	-754.75	0.00%	8	51	5	-633.45	16.07%	1	98	0
9	-4023.94	3.05%	5	22	4	-1711.50	58.76%	60	84	3
10	-310.00	0.00%	4	19	4	-132.00	57.42%	2	21	1
11	-9.10	0.00%	1	10	10	-9.10	0.00%	2	25	1
12	-28.42	0.00%	7	42	5	-27.55	3.06%	21	22	5
13	-52.32	0.00%	6	41	6	-52.32	0.00%	25	80	3
14	-77.18	0.00%	15	106	6	-75.60	2.05%	2	24	1
15	-101.78	0.58%	20	165	5	-101.88	0.48%	23	80	5
16	-127.38	0.71%	19	142	6	-125.01	2.56%	22	40	4
17	-181.34	1.53%	15	124	5	-178.943	2.83%	20	22	5
18	-237.12	1.88%	2	19	1	**-239.41**	**-0.94%**	21	40	5
19	-9.30	0.00%	1	10	5	-9.30	0.00%	1	24	0
20	-31.89	0.00%	4	37	9	-31.89	0.00%	21	22	4
21	-63.16	0.00%	1	10	11	-63.16	0.00%	2	25	1
22	-97.42	0.00%	1	10	18	-97.42	0.00%	2	24	1
23	-136.07	0.00%	8	73	6	-136.07	0.00%	22	23	4
24	-177.58	0.00%	3	28	3	-177.58	0.00%	25	60	2
25	-268.39	0.00%	10	91	2	-268.18	0.08%	20	24	2
26	-366.64	0.00%	8	73	2	-366.64	0.00%	1	24	0
Summary		7.76%	6.9	51.9	5.9		151.71%	14.5	38.8	2.2

Table 2. Results of the base-case runs of TTM and TCM

The "Best value" columns in Table 2 show the objective value of the best solution found by TTM and TCM, and Gap is the percentage difference between the best known objective function value and the objective function value corresponding to the best solution found by each method. The "Iter to best" and "Solver calls" columns show how quickly in the solution process the best solution is obtained in terms of the number of major iterations and the number of solver calls. The "successes" columns show the

number of tunnel or pseudo-cut loop successes, as defined above. The last row in Table 2 shows the total gap and the average values for all the other measures.

The results obtained with these base runs of the TTM are encouraging, with 23 of the 26 problems solved to gaps of less than 1% and 21 solved to gaps of essentially zero. The total of the 26 gaps is 7.76%, a measure that we use later to compare algorithmic options. On the other hand, the TCM obtains lower quality results when compared to those of the TTM, with a gap sum of 151.71%. This gap sum includes three negative associated with the three new best known solutions found by TCM in the set of test problems (problems 4, 5 and 18). These results indicate that TTM is a method that exhibits a more robust behavior than TCM, when defining robustness as the ability to yield high quality solutions consistently. A measure of robustness, for instance, may be given by the standard deviation of the gap values, which for the experiments reported in Table 2, is 0.74% for TTM and 17.21% for TCM. Regarding these measures, both methods compare well with Conopt's 547.32% total gap and 30.35% standard deviation.

Of the 5 problems (9 and 15-18) for which TTM obtained nonzero gaps, 4 of them (problems 15-18) are the largest of the EX8_6_1_n series. These problems have hundreds of local optima and are known to be very difficult for any global solver. The widely used multi-start solver OQNLP does not solve any of these 5 problems to gaps of less than 1%, as we discuss momentarily. Regarding computational efficiency, TTM finds its best solutions after roughly 1/3 of the major iteration budget of 20, averaging 6.9 major iterations to achieve its best solutions. The best solutions are found in 10 or fewer iterations in 20 of 26 problems. The average solver calls to best is 51.9, which is less than 25% of the maximum number of solver calls set at 221. An average of 6 tunneling loops out of 20 achieve an objective value at least as good as the best value discovered thus far, although some of these successes are revisits to the best known solution. Thus, most tunneling loops do not improve the best solution, but in many cases starting the original problem from the final points of these loops does improve it.

The TCM obtains the best known solutions in 13 out of 26 instances and it is able to improve the best known solution in three instances. However, 8 problems resulted in gaps larger than 1%. Surprisingly, the largest gaps are obtained in problems belonging to the Ex2_1_7_n series, which the literature does not consider as hard as problems in the other series included in this set of test problems. We now examine, in more detail, the performance of TTM, which, as pointed out above, seems to be the more robust of the two methods that we have developed.

5.2. Comparison of TTM with OQNLP

Table 3 shows the results of applying the multi-start OQNLP solver to problems 9 and 15-18, for which TTM failed to match the objective function value of the best known solution. To this set, we add problem 14, which has also been difficulty for OQNLP. After applying OQNLP with its default values for termination criteria and tolerances were used (i.e., 1000 candidate starting points with 200 in stage 1), we use the resulting best solution to start a TTM search.

Table 3 shows that OQNLP yields positive gaps for all problems (see "Initial gap" column) and for only one of the problems (number 9) it is able to find a solution with a gap value under 1%. The sum of the gap values is 15.65% which is more than twice as large as TTM's gap sum of 7.76% for these problems when executing both searches from the same initial points. When starting TTM from the final OQNLP solution solves 2 problems to zero gaps, leaves 2 others with gaps less than 1%, and achieves a gap sum of 4.77%, as shown in Table 3

Problem number	OQNLP obj	Initial gap	TTM obj	TTM gap	Major iter	Solver calls	Tunnel successes	Failures /calls
9	-4118.725	0.76%	-4140.449	0.24%	5	42	1	0 /36
14	-76.21	1.25%	-77.177	0.00%	1	4	1	32 / 47
15	-98.949	3.34%	-102.373	0.00%	15	126	2	30 /36
16	-123.827	3.48%	-128.097	0.15%	10	79	3	37 /45
17	-178.382	3.14%	-179.773	2.38%	1	4	1	33/ 45
18	-232.795	3.67%	-236.845	2.00%	2	13	2	40 /46
Total		15.65%		4.77%	5.7	44.7	1.7	

Table 3. Results of first applying OQNLP and then TTM to a selected subset of problems

The last column of Table 3 shows that when OQNLP is applied to the EX8_6_1_n problems (numbers 14 to 18), most of the Conopt calls fail to find a local optimum. This is because many of Conopt's function evaluations are at points where there are domain violations — the denominator of one or more objective terms vanishes. Our GAMS model of these problems includes nonlinear constraints which require these denominators to exceed some small positive value, but the starting points used by OQNLP may not satisfy these constraints. These points are generated by the OptQuest scatter search procedure (Ugray et al. 2007), and there are options to force OptQuest to generate points which satisfy linear constraints, but no such options exist within OptQuest for nonlinear constraints. Since all points generated by TTM are feasible, domain violations are never a problem, and all TTM Conopt calls for this family of problems end with a local optimum. This illustrates an important advantage of TTM over any multi-start method whose starting points are not guaranteed to be feasible.

5.3 Changing the Random Perturbation

In steps 9-11 of the TTM pseudo-code in Figure 2, a random perturbation vector r is generated, a multiple of r is added to the current local solution s, and the feasible point closest to the perturbed point is found by solving the projection problem in step 12. Since each TTM subproblem may be nonconvex, a local solver applied to it may find a final solution which depends on its starting point. Hence TTM's progress is sensitive to the values of r, and each run of TTM that uses a different sequence of random values for r will generally differ from the others in the objective values found after each major and minor iteration, and in the best value found overall. The results shown in Tables 2 and 3 all used the same seed for the GAMS random number generator, so that the same results would be achieved if the problem were solved several times.

Table 4 shows the results of changing the random number seed to 5 values, all different from each other and from the value used in Table 2, on two of the problems (9 and 15) solved by TTM in Table 2 to positive gaps.

Problem number	Run	Final Obj	Gap	Gap < 1% ?	Major iter to best	Solver calls	Tunnel successes
9	1	-4150.4101	0.00%	1	5	28	4
	2	-4150.4101	0.00%	1	11	82	3
	3	-4150.4101	0.00%	1	5	26	6
	4	-4023.9400	3.05%	0	6	29	5
	5	-4023.9400	3.05%	0	13	100	4
15	1	-101.8780	0.48%	1	20	163	7
	2	-100.7440	1.59%	0	15	118	5
	3	-101.7750	0.58%	1	11	78	5
	4	-101.8020	0.56%	1	2	9	2
	5	-101.8780	0.48%	1	5	34	2

Table 4. 5 runs of 2 problems with different random number seeds

Three of the 5 runs for problem 9 achieve the best known solution, while two end with the same best objective value as found originally in Table 2. Four of the Final Obj values for problem 15 improve upon the Table 2 value, with one slightly worse. Thus, if there is enough run time available, TTM can achieve improved results simply by making multiple runs with different seeds. As in Table 2, the last three columns in Table 4 show the number of major iterations to the best solution, the numbers of solver calls to the best solution and the number of tunnel successes, respectively.

5.4 Increasing the Number of Major Iterations

We have observed that TTM often finds better results if the major iteration limit is increased. Table 5 shows the outcomes of increasing the major iteration limit to 40 on the 5 problems that yielded positive gaps in the experiments reported in Table 2. We employ the same random seed used in all the Table 2 runs.

Problem number	Final Obj	Gap	Improved?	Major iter	Solver calls	Tunnel successes
9	-4105.278	1.09%	Y	37	304	5
15	-101.775	0.58%	N	20	165	5
16	-127.442	0.66%	Y	22	163	7
17	-181.418	1.49%	Y	21	178	6
18	-239.252	1.00%	Y	26	235	2
Total		4.82%				

Table 5. Solving 5 Positive Gap problems allowing 40 major iterations

As shown in the “Improved?” column of Table 5, four of the 5 final objective values improve when increasing the number of major iterations from 20 to 40. The sum of the gaps is reduced from 7.76% to 4.82%.

5.5 Combining TTM and TCM

We considered the combination of our two proposed approaches. Specifically, the TTM+TCM combination consists of running TTM and then applying TCM from the best solution found by TTM. Symmetrically, in the TCM+TTM we first apply TCM and then TTM. Table 6 reports the results of TTM, TTM+TCM and TCM+TTM on the 6 problems in Table 5.

	TTM		TTM+TCM		TCM+TTM	
Problem no.	**Obj**	**Gap**	**Obj**	**Gap**	**Obj**	**Gap**
9	-17.000	0.00%	-17.000	0.00%	-17.000	0.00%
15	-101.775	0.58%	-102.373	0.00%	-101.199	1.14%
16	-127.377	0.71%	-127.762	0.41%	-126.927	1.06%
17	-181.341	1.53%	-181.341	1.53%	-181.650	1.36%
18	-237.118	1.88%	-237.118	1.88%	-236.757	2.03%
Total		4.71%		3.82%		5.60%

Table 6. Results of combining TTM and TCM

Table 6 shows that the combination of TTM+TCM is able to improve upon the original TTM resulting in a gap sum of 3.82 % which compares favorably with the 4.71 % obtained by the TTM alone. On the other hand, the combination TCM+TTM does not perform well on average but yields the best result for problem 17 when compared to any of the other methods, including TTM with 40 major iterations (as shown in Table 5).

6. Conclusions and Future Work

We have described an extension of a pure tunneling method, which we have denoted as the tabu tunneling method (TTM) and an adaptation of the pseudo-cut strategy denoted as tabu cutting method (TCM). The main idea of both approaches is the addition of a short term memory structure that is typical to tabu search procedures. The testing described here indicates that the resulting procedures are promising: (1) the TTM is competitive with the best existing multi-start procedure for smooth constrained NLP's, OQNLP, and (2) the TCM improves the local search based methods, such as Conopt, and although it found two new best solutions, its average performance is inferior to TTM. The testing has been performed using GAMS, however, further testing on a larger problem set is needed, and this requires that both approaches be implemented in a more general language.

A direction for future work relates to the use of filters in order to determine points from which to apply the local solver to the original problem. Currently a solution found by solving a tunneling subproblem is used as a starting point for the original problem only if its objective value is at least as good as the best found so far, or if the iteration limit *MaxIter* (currently 5) is reached, in which case the last tunneling solution is used. TTM might be even more efficient if additional criteria were applied to determine whether or not to solve the original problem from one or more of the tunneling loop solutions.

Following the ideas used in OQNLP, one might start from a tunneling solution whose true objective value was below a threshold — called the *merit filter* in Ugray et al. (2007) — and which was sufficiently far from any local solution of the original problem found so far (the *distance filter*). The merit filter criterion is applied in this version of TTM, with the threshold equal to the best objective value found thus far. However, a second more relaxed threshold could be used in conjunction with a distance filter, while exceeding the current threshold would terminate the tunneling loop unconditionally.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Ref. TIN2006-02696).

References

Alp, S., G. Ertek and S. Birbil (2006) "Application of the Cutting Stock Problem to a Construction Company: A Case Study," in *Proceedings of the 5th International Symposium on Intelligent Manufacturing Systems*, May 29-31, pp. 652-661.

April, J., M. Better, F. Glover, J. Kelly and M. Laguna (2006) "Enhancing Business Process Management with Simulation-Optimization," in *Proceedings of the 2006 Winter Simulations Conference*, L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto (eds.), pp. 642-649.

Better, M., F. Glover and M. Laguna (2007) "Advances in Analytics: Integrating Dynamic Data Mining with Simulation Optimization," *IBM Journal of Research and Development*, vol. 51, no. 3/4, pp. 477-487.

Conejo, A. J., E. Castillo, R. Minguez and R. Garcia-Bertrand (2006) *Decomposition Techniques in Mathematical Programming*, Springer, 541 pp.

Drud, A. (1994) "CONOPT—A Large-Scale GRG-Code," *INFORMS Journal on Computing*, vol. 6, no. 2, pp. 207-216.

Floudas, C.A., P.M. Pardalos, C.S. Adjiman, W.R. Esposito, Z. Gumus, S.T. Harding, J.L. Klepeis, C.A. Meyer and C.A. Schweiger (1999) *Handbook of Test Problems for Local and Global Optimization*, Springer, 484 pp.

Glover, F. (1989) "Tabu Search - Part I," *INFORMS Journal on Computing*, vol. 1, no. 3, pp. 190-206.

Glover, F. (2008) "Pseudo-Cut Strategies for Global Optimization," OptTek Systems, Inc., Technical Report.

Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers: Boston, ISBN 0-7923-9965-X, 408 pp.

Levy, A.V. and S. Gomez (1985) "The Tunneling Method Applied to Global Optimization," in *Numerical Optimization*, Boggs P. T., R. H. Byrd and R. B. Schnabel (eds.), SIAM, pp. 213-244.

Murray, W. and K-M. Ng (2002) "Algorithms for Global Optimization and Discrete Problems based on Methods for Local Optimization," in *Handbook of Global Optimization Volume 2,* P. M. Pardalos and H. E. Romeijn (eds.), Springer, pp. 87-114.

Smith, S. and L. Lasdon (1992) "Solving Large Nonlinear Programs Using GRG," *INFORMS Journal on Computing*, vol. 4, no. 1, pp. 2-15.

Ugray, Z., L. Lasdon, J. Plummer, F. Glover, J. Kelly and R. Martí (2007) "Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization," *INFORMS Journal on Computing*, vol. 19, no. 3, pp. 328-340.