Proceedings of GO 2005, pp. 1 – 5.

Variable Neighbourhood Search for the Global Optimization of Constrained NLPs

Leo Liberti,[1] and Milan Dražić[2]

[1] *DEI, Politecnico di Milano, P.zza L. da Vinci 32, 20133 Milano, Italy,* liberti@elet.polimi.it

[2] *Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia and Montenegro,* mdrazic@matf.bg.ac.yu

Abstract We report on the theory and implementation of a global optimization solver for general constrained nonlinear programming problems based on Variable Neighbourhood Search, and we give comparative computational results on several instances of continuous nonconvex problems. Compared to an efficient multi-start global optimization solver, the VNS solver proposed appears to be significantly faster.

Keywords: VNS, global optimization, nonconvex, constrained, NLP.

1. Introduction

This paper describes a Variable Neighbourhood Search (VNS) solver for the global solution of continuous constrained nonlinear programming problems (NLPs) in general form:

$$\left.\begin{array}{lrcl} \min\limits_{x\in\mathbb{R}^n} & & f(x) & \\ \text{s.t.} & l \leq & g(x) & \leq u \\ & x^L \leq & x & \leq x^U. \end{array}\right\} \tag{1}$$

In the above formulation, x are the problem variables. $f : \mathbb{R}^n \to \mathbb{R}$ is a possibly nonlinear function, $g : \mathbb{R}^n \to \mathbb{R}^m$ is a vector of m possibly nonlinear functions, $l, u \in \mathbb{R}^m$ are the constraint bounds (which may be set to $\pm\infty$ as needed), and $x^L, x^U \in \mathbb{R}^n$ are the variable bounds.

Previous work on Variable Neighbourhood Search applied to global optimization was restricted to box-constrained NLPs ($m = 0$ in the above formulation) [19]. To the best of our knowledge, a VNS solver for constrained global optimization targeted at problems in general form (1) has not been implemented yet. It is worth noting, however, that the box-constrained VNS solver described in [19] is currently being tested on a reformulation of constrained problems based on penalization of explicit constraints.

2. The Variable Neighbourhood Search algorithm

Variable Neighbourhood Search (VNS) is a relatively recent metaheuristic which relies on iteratively exploring neighbourhoods of growing size to identify better local optima [6–8]. More precisely, VNS escapes from the current local minimum x^* by initiating other local searches from starting points sampled from a neighbourhood of x^* which increases its size iteratively until a local minimum better than the current one is found. These steps are repeated until a given termination condition is met.

VNS has been applied to a wide variety of problems both from combinatorial and continuous optimization. Its early applications to continuous problems were based on a particular problem structure. In the continuous location-allocation problem the neighbourhoods are defined according to the meaning of problem variables (assignments of facilities to custmers, positioning of yet unassigned facilities and so on) [3]. In bilinearly constrained bilinear problems the neighbourhoods are defined in terms of the applicability of the successive linear programming approach, where the problem variables can be partitioned so that fixing the variables in either set yields a linear problem; more precisely, the neighbourhoods of size k are defined as the vertices of the LP polyhedra that are k pivots away from the current vertex [6]. However, none of the early applications of VNS to continuous problems was ever designed for solving problems in general form (1).

The first VNS algorithm targeted at problems with fewer structural requirements, namely, box-constrained NLPs, was given in [19] (the paper focuses on a particular class of box-constrained NLPs, but the proposed approach is general). Since the problem is assumed to be box-constrained, the neighbourhoods arise naturally as hyperrectangles of growing size centered at the current local minimum x^*.

Algorithm 1 The VNS algorithm.

0: Input: maximum number of neighbourhoods $k_{\max}$, number of local searches in each neighbourhood L.
loop
  Set $k \leftarrow 1$, pick random point $\tilde{x}$, perform a local search to find a local minimum x^*. (†)
  while $k \leq k_{\max}$ **do**
    Consider a neighbourhood $N_k(x^*)$ of x^* such that $\forall\ k > 1\ (N_k(x^*) \supset N_{k-1}(x^*))$.
    for $i = 1$ to L **do**
      Sample a random point $\tilde{x}$ from $N_k(x^*)$.
      Perform a local search from $\tilde{x}$ to find a local minimum x'. (†)
      If x' is better than x^*, set $x^* \leftarrow x'$, $k \leftarrow 0$ and exit the FOR loop.
    end for
    Set $k \leftarrow k + 1$.
    Verify termination condition; if true, exit.
  end while
end loop

In Algorithm 1, the termination condition can be based on CPU time, number of non-improving steps and so on.

We implemented Algorithm 1 so that steps (†), i.e. the local search phases, are carried out by an SQP algorithm which is capable of locally solving constrained NLPs.

The definition of the neighourhoods may vary. Consider hyperrectangles $H_k(x)$, centered at x and proportional to the hyperrectangle $x^L \leq x \leq x^U$ given by the original variable bounds, such that $H_{k-1}(x) \subset H_k(x)$ for each $k \leq k_{\max}$. Letting $N_k(x) = H_k(x)$, sampling becomes extremely easy. There is a danger, though, that sampled points will actually be inside $H_{k-1}(x)$, which had already been explored at the previous iteration. Even though the likelihood of this situation arising lessens as the dimension of the Euclidean space increases (since the higher the dimension, the higher the ratio of the volume of $H_k(x)$ to the volume of $H_{k-1}(x)$), we would like to make sure that the sampled points are outside $H_{k-1}(x)$.

Naturally, taking $N_k(x) = H_k(x) \backslash H_{k-1}(x)$ solves this particular difficulty. Sampling in $H_k(x) \backslash H_{k-1}(x)$ is not as straightforward as sampling in $H_k(x)$, however. Let τ be the affine map sending the hyperrectangle $x^L \leq x \leq x^U$ into the unit L_∞ ball (i.e., hypercube) B centered at 0. Let $r_k = \frac{k}{k_{\max}}$ be the radii of the balls B_k (centered at 0) such that $\tau(H_k(x)) = B_k$ for each $k \leq k_{\max}$. In order to sample a random vector $\tilde{x}$ in $B_k \backslash B_{k-1}$ we proceed as follows:

1 sample a random direction vector $d \in \mathbb{R}^n$;

2 normalize d (i.e., set $d \leftarrow \frac{d}{||d||_\infty}$);

3 sample a random radius $r \in [r_{k-1}, r_k]$ yielding a uniformly distributed point in the shell;

4 let $\tilde{x} = rd$.

Finally, of course, we set $\tilde{x} \leftarrow \tau^{-1}(\tilde{x})$. With this construction, we obtain $\tilde{x} \in H_k(x) \backslash H_{k-1}(x)$.

3. The implementation

The search space is defined as the hyperrectangle given by the set of variable ranges $x^L \leq x \leq x^U$. At first we pick a random point $\tilde{x}$ in the search space, we start a local search and we store the local optimum x^*. Then, until k does not exceed a pre-set $k_{\max}$, we iteratively select new starting points $\tilde{x}$ in an increasingly larger neighbourhood $N_k(x^*)$ and start new local searches from $\tilde{x}$ leading to local optima x'. As soon as we find a local optimum x' better than x^*, we update $x^* = x'$, re-set $k = 1$ and repeat. Otherwise the algorithm terminates.

For each $k \leq k_{\max}$ we consider hyperrectangles $H_k(x^*)$ proportional to $x^L \leq x \leq x^U$, centered at x^*, whose sides have been scaled by $\frac{k}{k_{\max}}$. More formally, let $H_k(x^*)$ be the hyperrectangle $y^L \leq x \leq y^U$ where, for all $i \leq n$,

$$\begin{aligned} y_i^L &= x_i^* - \frac{k}{k_{\max}}(x_i^* - x_i^L) \\ y_i^U &= x_i^* + \frac{k}{k_{\max}}(x_i^U - x_i^*). \end{aligned}$$

This construction forms a set of hyperrectangular "shells" centered at x^* and proportional to $x^L \leq x \leq x^U$. As has been mentioned above, we define each neighbourhood $N_k(x^*)$ as $H_k(x^*) \backslash H_{k-1}(x^*)$.

The main solver parameters control: the minimum neighbourhood size, the number of sampling points and local searches started in each neighbourhood (L in Algorithm 1), an ε tolerance to allow moving to a new x^* only when the improvement is sufficiently high, and the maximum CPU time allowed for the search.

The solver was coded within the *ooOPS* optimization software framework [17]. *ooOPS* allows global optimization solvers to be deployed quickly and efficiently by offering an API which is very rich in functionality. Solvers can call each other as black-box procedures; the solver library is still modest but growing, including SNOPT [5], the NAG library NLP solver [21], `lp_solve` as local solvers and sBB [12, 13], SobolOpt [10] and the VNS solver described in this paper as global solvers; a rich symbolic computation library is provided, whose capabilities extend to the symbolic computation of derivatives and automatic simplification of expressions, as well as to generating a convex relaxation of the problem at hand.

4. Computational results

In Table 1 we have collected a number of test instances of various constrained NLPs (small, medium and large sized, ordered by number of problem variables) and reported solution times of the VNS solver described in this paper versus those obtained with the SobolOpt solver [10], a Multi Level Single Linkage algorithm based on low-discrepancy Sobol' sequences sampling. In both cases, the local NLP solver used is SNOPT [5]. Both solvers managed to locate the global optima in all test instances. The instances were solved on a Pentium IV 2.66MHz CPU with 1GB RAM, running Linux. All parameters were set to their default values. $k_{\max}$ was set to 10 for most of the test instances, apart from the largest ones (the last four in Table 1), where it was set to 50.

The `griewank-2` instance is a modified Griewank function described in [18]. `sixhump` is a classic test function for global optimization algorithm. `sntoy` is the "toy problem" found in the SNOPT documentation [5]. `bilinear-eg3` was taken from [14]: it is a MINLP with 1 binary variable, reformulated to continuous subject to $x^2 = x$. `haverly`, `ben-tal4`, `example4` and `foulds3` are instances of bilinear multi-quality blending problems arising from the oil industry. `kissing-24_4` solves the Kissing Number problem in 4 dimensions. `lavor50` and `more4` are instances from the Molecular Distance Geometry Problem; although `more4` has more atoms and nonlinear terms, the particular way in which the instance `lavor4` was generated makes it harder to solve.

Instance	From	N	C	T	VNS	SobolOpt
`griewank-2`	[18] (Test function)	2	0	4	0.005^*	1.03
`sixhump`	[22] (Classic test function)	2	0	6	0.006	0.001^*
`sntoy`	[5] (SNOPT test problem)	4	3	5	7.01	2.64^*
`bilinear-eg3`	[14] (MINLP test problem)	6	5	15	0.09	0.03^*
`haverly`	[9] (Haverly's pooling problem)	9	8	6	0.01^*	0.04
`ben-tal4`	[2] (Blending problem)	10	8	6	0.01^*	0.44
`example4`	[1] (Blending problem)	26	35	48	0.62^*	1.51
`kissing-24_4`	[16] (Kissing Number problem)	97	300	1200	51.92^*	213.70
`lavor50`	[11] (Molecular conformation)	150	0	16932	707.15^*	3153.02
`foulds3`	[4] (Blending problem)	168	48	136	0.46^*	12.65
`more4`	[20] (Molecular conformation)	192	0	45288	418.818^*	2903.38

Table 1. Computational results comparing the VNS with the SobolOpt solvers in *ooOPS*. User CPU timings are in seconds. Values marked with * denote the best timings.

5. Conclusion

We presented a new global optimization solver for constrained NLPs based on the VNS algorithm. The computational comparison with an efficient Multi-Level Single Linkage (MLSL) algorithm called SobolOpt seems to show that in general VNS performs better than MLSL. Future work will be two-fold: on one hand, another VNS global solver based on box-constrained reformulation of constrained NLPs based on penalization of explicit constraints is being tested so that more relevant comparative computational data can be gathered. On the other hand, we plan to take into account the explicit constraint structure of the problem even in the global phase, and not only in the local phase as is currently the case. The latter development should be beneficial particularly for those problems for which the local solver has trouble finding a feasible starting point.

Acknowledgments

We are particularly thankful to Dr. Sergei Kucherenko for providing the code for the SobolOpt MLSL solver.

References

[1] N. Adhya, M. Tawarmalani, and N.V. Sahinidis. A lagrangian approach to the pooling problem. *Industrial and Engineering Chemistry Research*, 38:1956–1972, 1999.

[2] A. Ben-Tal, G. Eiger, and V. Gershovitz. Global minimization by reducing the duality gap. *Mathematical Programming*, 63:193–212, 1994.

[3] J. Brimberg and N. Mladenović. A variable neighbourhood algorithm for solving the continuous location-allocation problem. *Studies in Location Analysis*, 10:1–12, 1996.

[4] L.R. Foulds, D. Haughland, and K. Jornsten. A bilinear approach to the pooling problem. *Optimization*, 24:165–180, 1992.

[5] P.E. Gill. *User's Guide for SNOPT 5.3*. Systems Optimization Laboratory, Department of EESOR, Stanford University, California, February 1999.

[6] P. Hansen and N. Mladenović. Variable neighbourhood search: Principles and applications. *European Journal of Operations Research*, 130:449–467, 2001.

[7] P. Hansen and N. Mladenović. Variable neighbourhood search. In P. Pardalos and M. Resende, editors, *Handbook of Applied Optimization*, Oxford, 2002. Oxford University Press.

[8] P. Hansen and N. Mladenović. Variable neighbourhood search. In F.W. Glover and G.A. Kochenberger, editors, *Handbook of Metaheuristics*, Dordrecht, 2003. Kluwer.

[9] C.A. Haverly. Studies of the behaviour of recursion for the pooling problem. *ACM SIGMAP Bulletin*, 25:19–28, 1978.

[10] S. Kucherenko and Yu. Sytsko. Application of deterministic low-discrepancy sequences to nonlinear global optimization problems. *Computational Optimization and Applications*, 30(3):297–318, 2004.

[11] C. Lavor. On generating instances for the molecular distance geometry problem. In Liberti and Maculan [15], pages 405–414.

[12] L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College London, UK, March 2004.

[13] L. Liberti. Writing global optimization software. In Liberti and Maculan [15], pages 211–262.

[14] L. Liberti. Linearity embedded in nonconvex programs. *Journal of Global Optimization*, (to appear) 2004.

[15] L. Liberti and N. Maculan, editors. *Global Optimization: from Theory to Implementation*. Springer, Berlin, (to appear).

[16] L. Liberti, N. Maculan, and S. Kucherenko. The kissing number problem: a new result from global optimization. In L. Liberti and F. Maffioli, editors, *CTW04 Workshop on Graphs and Combinatorial Optimization*, volume 17 of *Electronic Notes in Discrete Mathematics*, pages 203–207, Amsterdam, 2004. Elsevier.

[17] L. Liberti, P. Tsiakis, B. Keeping, and C.C. Pantelides. *ooOPS*. Centre for Process Systems Engineering, Chemical Engineering Department, Imperial College, London, UK, 2001.

[18] M. Locatelli. A note on the griewank test function. *Journal of Global Optimization*, 25:169–174, 2003.

[19] N. Mladenović, J. Petrović, V. Kovačević-Vujčić, and M. Čangalović. Solving a spread-spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operations Research*, 151:389–399, 2003.

[20] J.J. Moré and Z. Wu. Global continuation for distance geometry problems. *SIAM Journal on Optimization*, 7:814–836, 1997.

[21] Numerical Algorithms Group. *NAG Fortran Library Manual Mark 11*. 1984.

[22] E.M.B. Smith. *On the Optimal Design of Continuous Processes*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.