MATHOPTIMIZER PROFESSIONAL *Key Features and Illustrative Applications*

János D. Pintér[1] and Frank J. Kampas[2]
[1] Pintér Consulting Services, Inc., Halifax, Nova Scotia, Canada
jdpinter@hfx.eastlink.ca http://www.pinterconsulting.com
[2] WAM Systems, Inc., Plymouth Meeting, PA, USA
fkampas@wamsystems.com http://www.wamsystems.com

To appear in:

Liberti, L., and Macaulan, N., Editors

Global Optimization: From Theory to Implementation

Springer Science + Business Media, New York, 2005.

Chapter

MATHOPTIMIZER PROFESSIONAL
Key Features and Illustrative Applications

János D. Pintér[1] and Frank J. Kampas[2]
[1] Pintér Consulting Services, Inc., Halifax, Nova Scotia, Canada
jdpinter@hfx.eastlink.ca http://www.pinterconsulting.com
[2] WAM Systems, Inc., Plymouth Meeting, PA, USA
fkampas@wamsystems.com http://www.wamsystems.com

Abstract: Integrated scientific-technical computing (ISTC) environments play an increasing role in advanced systems modeling and optimization. MathOptimizer Professional (MOP) has been recently developed to solve nonlinear optimization problems formulated in the ISTC system *Mathematica*. We introduce this software package, and review its key functionality and options. MOP is then used to solve illustrative circle packing problems, including both well-frequented models and a new (more difficult) model-class.

Key words: Integrated computing systems; *Mathematica*; global optimization; LGO solver suite; MathOptimizer Professional; uniform and arbitrary size circle packings; illustrative results.

1. INTRODUCTION

Operations Research (O.R.) provides a consistent quantitative framework and techniques, to assist analysts and decision-makers in finding "good" (feasible) or "best" (optimal) solutions in a large variety of contexts. For an overview of prominent O.R. application areas, consult e.g. the 50th anniversary issue of the journal *Operations Research* (2002).

A formal procedure aimed at finding optimized decisions consists of the following key steps.

- Conceptual description of the decision problem at a suitable level of abstraction that retains all essential attributes, but omits secondary details and circumstances.

- Development of a quantitative model that captures the key elements of the decision problem, in terms of decision variables and functional relationships among them.
- Development and/or adaptation of an algorithmic solution procedure, in order to explore the set of feasible solutions, and to select the best decision.
- Numerical solution of the model and its verification; interpretation and summary of results.
- Posterior analysis and implementation of the decision(s) selected.

The problems tackled by O.R. are often so complex that the correct model and solution procedure may not be clear at the beginning. Therefore, decision makers often must carry out the steps outlined above in an iterative fashion. The analyst repeatedly modifies and refines the model formulation and solution procedure until the model captures the essence of the problem, is computationally tractable, and its numerical solution is applicable in the context of the problem studied.

These considerations make a strong case for using high-level, integrated software tools that can effectively assist in performing all related tasks in a unified framework. This point is particularly valid in modeling nonlinear systems, since their analysis may involve the evaluation of computationally intensive functions, visualization, animation, and so on.

Maple (Maplesoft, 2004a), *Mathematica* (Wolfram Research, 2004), and Matlab (MathWorks, 2004) are prominent, fully integrated scientific-technical computing systems. The capabilities and range of applications of these software products and related application packages are documented in software manuals, hundreds of books, and many thousands of articles and presentations. The current user base of ISTC systems is several million people worldwide.

A concise list of the most significant features and capabilities of ISTC environments includes the following (note that each feature listed below is currently supported by at least one – and sometimes by all – of the three ISTC systems mentioned):

- A broad range of simple and advanced computations with high – or even with arbitrarily high, adjustable – precision
- Support for symbolic calculations
- Extensive set of readily available functions, from programming language standards to special functions and general-purpose, complete numerical procedures (examples of the latter are integration routines, differential equation solvers, and numerical optimization routines)
- Context-specific "point and click" (essentially syntax-free) operations via GUI elements that help to execute various tasks
- Support for concise, transparent code development and maintenance

- Support for several programming styles (procedural, functional, and rule-based paradigms)
- Full programmability (i.e., extendibility by adding new functionality)
- Posterior analysis and implementation of the decision(s) selected
- Advanced technical documentation, desktop publishing, and presentation features
- Interactive and multimedia tools (in-situ evaluation, visualization, animation, sound)
- Built-in, fully integrated help system that includes portable application examples
- Automatic code generation from a given ISTC language to more "traditional, lower-level" programming languages (such as Basic, C, Fortran, Java, and so on)
- Automatic conversions of ISTC system documents to tex, html, xml, ps, pdf (and possibly other) file formats
- Direct links to external application packages, to other software products, and to the Internet
- Portability across a broad range of hardware platforms and operating systems (such as e.g., Windows, Macintosh, Linux, and Unix versions).

Many of these features can be effectively used during the various stages of developing O.R. applications. In particular, data analysis, model formulation, solution strategy and algorithm development, numerical solution, and project documentation can all be put together from the beginning – even in a single unified work document, if the developer wishes to do so. Hence, ISTC environments can increasingly provide a "one-stop" solution, to meet a broad range of needs of researchers, educators and students. This tendency has been receiving growing attention also in the O.R. community: for example, *Mathematica* has been recently reviewed in *ORMS Today* (Sodhi, 2003).

We emphasize here that, although all modeling and computational environments – from Excel spreadsheets, through optimization-specific algebraic modeling languages (such as AIMMS, AMPL, GAMS, LINGO, MPL, and others) to the more general-purpose computing (ISTC) systems – may have a lower program execution speed when compared to a compiled "pure number crunching" system, the overall application development time can often be massively reduced by using higher-level systems, especially when development can be started from scratch. It is instructive to recall in this context the debate that surrounded the early development of programming languages – such as Algol, Basic, C, Fortran, Pascal, etc. – as opposed to machine-level assembly programming.

Within the broad category of modeling and optimization problems, we see particularly strong application potentials for ISTC systems in studying nonlinear systems. For discussions of nonlinear system models and a broad range of their applications, consult e.g. Aris (1999), Bazaraa, Sherali, and Shetty (1993), Beltrami (1993), Bertsekas (1999), Bracken and McCormick (1968), Chong and Zak (2001), Diwekar (2003), Edgar, Himmelblau and Lasdon (2001), Gershenfeld (1999), Grossmann (1996), Hansen and Jørgensen (1991), Hillier and Lieberman (2005), Kampas and Pintér (2005), Murray (1983), Papalambros and Wilde (2000), Pardalos, Shalloway, and Xue (1996), Parlar (2000), Pearson (1986), Pintér (1996, 2005), Rich (1973), Schittkowski (2002), Tawarmalani and Sahinidis (2002), Wilson, Turcotte and Halpern (2003), Zabinsky (2003), and Zwillinger (1989).

2. GLOBAL OPTIMIZATION

As the books listed above well illustrate, nonlinearity is literally ubiquitous in the development (and modeling) of objects, formations and processes, including also living organisms of all scales, as well as various man-made systems. Decision-making (optimization) models that incorporate such a nonlinear system description frequently lead to complex problems that may or provably do have multiple – local and global – optima. The objective of global optimization (GO) is to find the "absolutely best" solution of nonlinear optimization models under such circumstances.

In order to formalize the general global optimization paradigm considered here, we shall use the following notation:

- x decision vector, an element of the real Euclidean n-space $\boldsymbol{R^n}$
- $f(x)$ objective function, f: $\boldsymbol{R^n} \to \boldsymbol{R^1}$
- D non-empty set of admissible decisions.

  The set D is defined by
- xl, xu explicit, finite bounds of x (an embedding "box" in $\boldsymbol{R^n}$)
- $g(x)$ m-vector of constraint functions, g: $\boldsymbol{R^n} \to \boldsymbol{R^m}$.

Applying the notation given above, the GO model can be stated as

$$(1) \quad \min f(x) \qquad x \in D := \{x: \quad xl \le x \le xu \quad g(x) \le 0\}.$$

Note that in (1) all inequalities are interpreted component-wise. Under fairly general analytical conditions, the GO model (1) has a global solution (set). (For example, if D is non-empty, and f, g are continuous functions, then the model has a non-empty set of global solutions X^*). Note that – although we know that X^* exists – if we use "traditional" local scope optimization strategies, then – depending on the starting point of the search – we will find only the corresponding locally optimal solution. In order to find – i.e., to properly approximate) – the "true" solution, a genuine global scope search effort is needed.

Global optimization is of great theoretical and practical importance, with significant existing and prospective applications. As of today (2004), a few hundred books, thousands of articles and dozens of web sites are devoted to the subject. For detailed discussions of the most prominent GO model types and solution approaches, consult, for example, Horst and Pardalos (1995), and Pardalos and Romeijn (2002), visit also Neumaier (2004). Among the earlier cited books on nonlinear systems models and optimization, e.g. Grossmann (1996), Kampas and Pintér (2005), Pardalos, Shalloway and Xue (1996), Pintér (1996, 2005), Tawarmalani and Sahinidis (2002), Zabinsky (2003) – as well as many others – discuss nonlinear models and real-world applications which require a global solution approach.

3. LGO SOLVER SUITE

The LGO (Lipschitz Global Optimizer) software package serves to find – i.e., to numerically approximate – global and local solutions to nonlinear optimization models. The current LGO implementation incorporates the following solver modules:

- Branch-and-bound global search method (BB)
- Global adaptive random search (single-start) (GARS)
- Multi-start based global random search (MS)
- Constrained local search (LS) by the reduced gradient method

All three global search methods will generate one (BB, GARS) or several (MS) approximations of the global optimizer point(s), before LGO is automatically switched to local search. The LS option can also be used in stand-alone mode, when started from a user-supplied initial point.

For theoretical details of the underlying global search methodology, consult Pintér (1996). The LS approach used is discussed in numerous textbooks: see for instance, Edgar et al. (2001). The LGO software system itself has been discussed in books and articles: consult e.g., Pintér, 2001, 2002, 2004, and the peer review by Benson and Sun, 2000. Therefore here we provide only a brief summary of the solver components. (Since LGO is a commercial software product, many of the implementation details will be omitted.)

The BB module is based on a theoretically established (rigorous) global optimization approach. BB combines set partition steps with deterministic and randomized sampling: this combination also enables a statistical bounding procedure. Note, however, the program runtimes can be expected to grow fast(er) for higher-dimensional and more difficult models, if we want to find a close approximation of the global solution solely by BB. (A similar comment applies also to all other theoretically rigorous global search methods.)

Pure random search is a very simple, "folklore" approach to global optimization that converges to the global solution (set) with probability 1. GARS is an improvement over that passive search approach in the sense that it adaptively attempts to focus the global search effort in the region which – on the basis of the actual sample results – is estimated to contain the global solution point (or, in general, one of these points).

Multi-start (MS) based global search applies a similar search strategy to GARS; however, the total sampling effort is distributed among several searches. Each of these leads to a "promising" starting point for subsequent local search. Typically, this approach takes the most computational effort (due to its multiple local searches); however – especially in more difficult models – it often finds the best numerical solution (Pintér, 2003).

In all global search modes an exact penalty (merit) function serves to aggregate the objective and constraint functions. Obviously, this assumes that the model functions are "acceptably" scaled. (A constraint penalty parameter can be adjusted via an LGO solver options file, to assist scaling.)

Ideally – and also in many actual model-instances – all three global search methods will give the same answer, except perhaps small numerical differences. In practice, especially when solving more difficult problems, the LGO user may wish to try all three global search options (in conjunction with the subsequent local search), to see which method gives the best results.

The local search (LS) component is based on the generalized reduced gradient (GRG) algorithm. In general, this search strategy is "only" locally convergent: therefore its use is recommended following one of the global searches, except in local optimization contexts. The application of LS, as a rule, results in a solution that is feasible and satisfies the Karush-Kuhn-Tucker local optimality conditions.

The solver suite approach – based on three different global solvers – supports the robust and efficient numerical solution of nonlinear models. The entirely derivative-free methods implemented in LGO enable the handling of merely computable model functions: this is of particular relevance with respect to applications, in which higher order (gradient, Hessian, etc.) information is impossible, difficult, or too costly to obtain. LGO can be used even to handle "black box" models provided (only) as object files, dynamic link libraries, or executable programs.

The LGO solver suite is currently available for C and Fortran compiler platforms, with customized links to Excel, GAMS, Maple, *Mathematica* and

Matlab (via TOMLAB). For specific descriptions of the versions not discussed here, see e.g. Frontline Systems and Pintér Consulting Services (2001), GAMS Development Corporation and Pintér Consulting Services (2003), Maplesoft (2004b), TOMLAB Optimization Inc. and Pintér Consulting Services (2004). LGO is also offered – in a demo (size-limited) version – with the latest edition of the classical O.R. textbook by Hillier and Lieberman (2005). LGO, in its various implementations, has been used in commercial applications, as well as in a variety of research and educational environments for more than a decade.

4. MATHOPTIMIZER PROFESSIONAL

The MathOptimizer Professional software product is based on an external LGO solver implementation that is seamlessly linked to the *Mathematica* platform. In other words, MathOptimizer Professional offers a combination of *Mathematica*'s sophisticated application development tools with core LGO solver functionality. This leads to a numerical performance that – in terms of both solution quality and solver speed – is comparable to other (compiler-based or optimization modeling language-related) LGO implementations, especially when models are more difficult and/or computationally intensive. In this section, we review the key features of MOP. Further details and an extensive list of practically motivated examples are discussed in the user manual (Pintér and Kampas, 2003), as well as in our forthcoming book (Kampas and Pintér, 2005).

The functionality of MOP can be summarized by the following steps:

- Optimization model formulation, in a *Mathematica* document (notebook)
- Automatic export and translation of the model into C or Fortran code
- Compilation of the generated code into a dynamic link library (DLL)
- Call to the external LGO engine: the latter is a "ready-made" executable program that is now linked to the model-dependent DLL
- Automatic model solution and report generation by LGO
- Import and display of results into the calling *Mathematica* notebook.

We refer to the approximately 150-page (printed) manual for further details.

It should be noted that the approach outlined supports automatically "only" the solution of models defined by *Mathematica* functions that can be directly converted into (C or Fortran) program code. This, however, still allows the handling of a fairly broad range of continuous nonlinear optimization models (including, of course, all models that could be directly written in C or Fortran). Other implementations (with extended functionality) are also possible.

One “side benefit” of using MOP is that models built in *Mathematica* can be directly used to generate corresponding C or Fortran test models. This is particularly advantageous in case of larger model-instances.

MathOptimizer Professional (MOP) – and its solver function `callLGO` − is launched by the *Mathematica* statement

```
Needs["MathOptimizerPro`callLGO`"];
```

The basic functionality of **`callLGO`** can be queried by the following *Mathematica* statement: see the auto-generated reply immediately below. (The format of this reply is slightly edited for the present purposes.)

```
?callLGO

callLGO[obj_, cons_List, varswithbounds_List,
opts___]:
obj is the objective function,
cons is a list of the constraints,
varswithbounds are the variables and their bounds in
the format {{variable, lower bound, initial value for
local search, upper bound}...} or {variable, lower
bound, upper bound}...}.

Function return is the value of the objective
function, a list of rules giving the solution, and
the maximum constraint violation.

See Options[callLGO] for the options and also see the
usage statements of the various options for their
possible values. For example, enter ?Method for the
possible settings of the Method option.
```

Table 1 (below) summarizes the current callLGO option list, with added notes. All options can be changed by users, following MOP specifications.

Option Name and Default Settings, with Additional Notes

ShowSummary→False	Display (or not) LGO report file
Method→MSLS	Alternatives: BBLS, GARSLS, LS
MaxEvals→ProblemDetermined	Global search effort, set by default to 1000*($n + m$); (global search phase stopping criterion)
MaxNoImprov→ProblemDetermined	Global search effort without sufficient improvement, set by default to 200*($n + m$) (global stopping criterion)
PenaltyMul→1	Penalty multiplier

ModelName→LGO Model	Model-dependent name (can be chosen by user)
DllCompiler→BC	Supported compilers: Borland C, Lahey Fortran, Microsoft C, Salford Fortran
ShowLGOInputs→False	Display (or not) LGO input files
LGORandomSeed→0	Set internally (can be reset by user)
TimeLimit→300	Seconds (global search phase stopping criterion)
TOBJFGL→ –1000000	Target objective function value in global search phase (global search phase stopping criterion)
TOBJFL→ –1000000	Target objective function value in local search phase (local search phase stopping criterion)
MFPI→10^{-6}	Merit function precision improvement tolerance (local search phase stopping criterion)
CVT→10^{-6}	Accepted constraint violation tolerance (local search phase stopping criterion)
KTT→10^{-6}	Kuhn-Tucker condition tolerance (local search phase stopping criterion)

Table 1. MathOptimizer Professional: callLGO options

As indicated above, **`callLGO`** is activated by a statement of the form

`callLGO[f,{g},{x,xl,xn,xu}, options]`.

Here the notations **`f`**, **`g`**, **`x`**, **`xl`** and **`xu`** directly correspond to the symbols defined in (1). In addition, **`xn`** (**`xl`**≤**`xn`**≤**`xu`**) is a user-supplied nominal solution – or a default setting, if **`xn`** is absent – that is used by LGO in its initial local search mode; finally, **`options`** denotes the calling parameters of the function **`callLGO`**.

The following simple example serves to illustrate the basic MOP functionality, as it appears to the user in default mode. Consider the model

$$\mathit{min}\ x^2 + 2\,y^2 \quad x + y \geq 1 \quad -2 \leq x \leq 2 \quad -2 \leq y \leq 2.$$

This optimization problem is solved by the next *Mathematica* statement that leads to the answer shown immediately below:

```
callLGO[x^2 + 2*y^2,
{x + y >= 1}, {{x, (-2, 0, 2}, {y, -2, 0, 2}}]
```

The answer received is a *Mathematica* list (as shown by the curly braces and comma separators):

```
{0.6666666666666666, {x -> 0.6666666667,
y -> 0.3333333333}, 5.551115123125783*^-17}
```

Here the first list element is the optimal objective function value found, followed by the list of corresponding variable assignments, and the maximal constraint violation at the solution point. (More details are shown automatically in the generated LGO report that can also be displayed in the notebook.) An extensive set of interesting GO challenges and practically motivated numerical examples are discussed also in the MOP User Guide.

In the numerical examples discussed here *Mathematica* versions 5.01 or 5.1 are used in conjunction with the Microsoft Visual C/C++ (MSVC, version 6.0) or the Salford FORTRAN 95 (FTN95) compiler. Furthermore, in all cases Pentium 4 processor based machines running under Microsoft Windows XP Professional version are used. Let us also note here that the RAM requirements of MOP *per se* are rather modest, at least for small or mid-size models; e.g., a personal computer with 256 Mb RAM is certainly adequate to handle MOP models with up to (at least) 1000 variables and 1000 constraints. In principle, arbitrarily large models can be handled using virtual memory, given sufficient hard drive space and time.

5. ILLUSTRATIVE APPLICATIONS: SOLVING SPHERE PACKING MODELS

Object packing models are aimed at finding the best non-overlapping arrangement of a set of objects in a container object. This general modeling paradigm can be specified in many ways, leading to interesting – and typically quite difficult – models. In addition to a more theoretical interest directed towards specific, analytically tractable problem-instances, there is also an obvious practical motivation to solve packing models.

In our recent and ongoing studies, we have found that this general model-class can be used to test and benchmark global optimization software. Several special cases and model-instances will be discussed below, to illustrate the potentials of numerical global optimization in solving object packing problems.

5.1 Packing Identical Size Circles in the Unit Square

The problem can be stated as follows: given the unit square and a positive integer *k*, find the maximal radius *r* of *k* identical circles that can be placed into the square, without overlaps. (Evidently, $r=r(k)$.)

This problem, as well as some other related circle packing models, has become a fascinating subject for both professional researchers and amateurs, at least in recent decades. There exists a significant body of literature (books, articles, dissertations, and web sites) discussing uniform circle packings. This information available includes proofs (only) for a small number of special cases (namely, for k=2,…,9, 14, 16, 25, and 36 circles), or computer aided optimality proofs, with guaranteed bounds (for up to k=30). In many other cases, only "best known" constructions are known.

To illustrate some rigorous bounding results, Csendes and his colleagues have applied interval arithmetic based GO methodology to prove bounds for best circle packings. The websites of Csendes (2004) and Markót (2004) list their related publications.

Referring to the general case, Graham noted in an interview (Albers, 1996) that he does not expect to know the true (i.e., proven optimal) solution for placing 1000 equal size circles in the unit square. The reason for his "learned skepticism" is that there is no unifying theory, and that intuition may fail: for example, in the k=49-circle case the seemingly obvious "seven-by-seven" configuration is not optimal, consult e.g. Specht (2004).

Without going into further details regarding this area of research, let us mention the thesis of Melissen (1997): he provides a detailed review of uniform circle packing model statements and key analytical results, with more than 350 topical references. The website of Specht (2004) is another rich source of information related to uniform size circle packings (in the unit square, the unit circle, and in rectangles): the site also includes references. We will compare our numerical results to those listed at this website as best (proved or postulated).

We also wish to emphasize that our sole purpose here is to illustrate the applicability of numerical global optimization (specifically, of LGO and thereby of MathOptimizer Professional) to difficult packing models, even without specifying or postulating any prior structure. More extensive studies should be based on more detailed and sophisticated modeling than what we are presenting here. We will cite only some of our existing numerical results: more details are available upon request, and will appear elsewhere.

Let us denote by $c_i=(x_i\ y_i)$ the center of circle $i=1,\ldots,k$: the coordinate pairs $(x_i\ y_i)$ and the radius r of these circles are the decision variables. The constraint that the circles i and j do not overlap means that the sum of their radii is less than the distance between the centers:

(2) $2r \le \| c_i - c_j \|$; here $\| c_i - c_j \| := [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}$ $1 \le i < j \le k$.

Note that each instance of the inequality in (2) is a non-convex constraint (since the norm function is convex). For k-circle configurations, we have $k(k\text{-}1)/2$ such constraints.

We shall consider the unit square that is centered at the origin. Additional constraints postulate that the circles are inside the enclosing square. These are derived from simple geometrical considerations; a possible formulation is:

(3) $| x_i | + r \le 0.5$ and $| y_i | + r \le 0.5$ $1 \le i \le k$.

This way, in addition to the non-convex constraints we have $2k$ non-linear convex constraints. (Alternative formulations are also possible, e.g. by replacing the latter constraints by linear ones. However, we have been using the constraints (3) in our numerical experiments.). Under the conditions (2)-(3), our objective is to

(4) *maximize r*.

We have developed a *Mathematica* model (a function) that is directly based on relations (2) to (4). This function is parameterized by the number of circles: therefore inserting a positive integer value k in the function provides the corresponding k-circle model instance. As noted earlier, MathOptimizer Professional automatically translates this *Mathematica* model to C or Fortran form, and then the external LGO solver is invoked to solve it. For illustration, the 20-circle solution found is shown below.

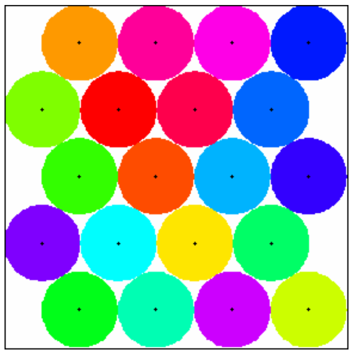

Figure 1. Packing 20 uniform size circles in the unit square

Note that this figure is directly imported from the *Mathematica* notebook document where the model formulation and all calculations have also been done. The optimized value of the circle radius $r=r(20)$ found by MOP is

$r \sim 0.1113823476$.

The radius found agrees well (to about 10^{-10} absolute precision) with the value 0.111382347512 posted at http://www.packomania.com. Note that such – arguably minor – imprecision can be due to several factors: one of the significant factors is that LGO applies central finite-difference based gradient approximation in its local search phase. This adds some error to that of standard floating point precision calculations.

The computer used to solve this example has a 3.2 GHz Pentium 4 processor and is running Windows XP; we used the Salford Fortran 95 compiler (Salford Software, 2004). The corresponding runtime is about 19 seconds. Note that runtimes may change slightly even when repeating the same run, due to hardware and OS status changes: however, the timing cited gives an impression of the MOP solver speed. (LGO per se runs faster, of course.) Recall that the 20-circle model instance of (2)-(4) has 41 decision variables, and 230 nonlinear constraints of which 190 are non-convex. Further detailed numerical results will appear e.g. in our forthcoming book (Kampas and Pintér, 2005).

5.2 Packing Identical Size Circles in the Unit Circle

This problem can be stated as follows: given the unit circle and a positive integer k, find the maximal radius $r=r(k)$ of k identical circles that can be placed into the circle, without overlaps.

Applying straightforward modifications of model (2)-(4), the adapted model can be written as

$$(5)\quad \begin{array}{ll} \textit{maximize } r & \\ 2r \le \| c_i - c_j \| & 1 \le i < j \le k \\ r + \| c_i \| \le 1 & 1 \le i \le k. \end{array}$$

As above, $\| c_i - c_j \| = [(x_i - x_j)^2 + (y_i - y_j)^2\,]^{1/2}$ and $\| c_i \| := (x_i^2 + y_i^2\,)^{1/2}$. The model (5) has $2k+1$ decision variables, k convex nonlinear constraints, and $k(k-1)/2$ non-convex constraints.

For illustration, we cite the solution of the 20-circle model instance (that has 41 decision variables and 210 constraints of which again 190 are non-convex). The next figure shows the configuration obtained.

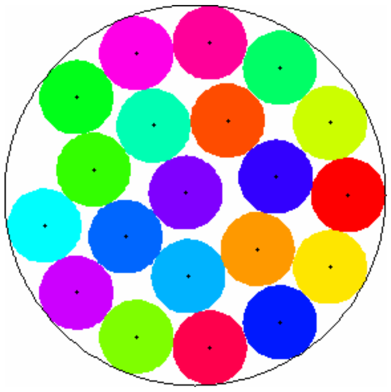

Figure 2. Packing 20 uniform size circles in the unit circle

The radius $r=r(20)$ found by MOP in this example equals 0.1952240114. This value agrees to at least 10^{-9} absolute precision with the best known result cited at www.packomania.com (0.1952240110...) The corresponding runtime is approximately 43 seconds, on the same computer as mentioned above.

5.3 Packing Non-Uniform Size Circles in an Optimized Circle

In this section, we introduce a new class of object packing models. Our objective is to find the minimal size circle that contains a given non-overlapping circle configuration that is made up by (in principle, arbitrary size) circles. To our best knowledge, this model has not been studied before by others in a GO setting: we also think that such models can be significantly more difficult than the more specific cases discussed in the preceding two sections.

We shall denote by r_i the radius of circle i for $i=1,\dots,k$. With a straightforward generalization of (5), we obtain the model

$$(6) \quad \textit{minimize } r$$
$$r_i + r_j \leq \| c_i - c_j \| \quad 1 \leq i < j \leq k.$$
$$r_i + \| c_i \| \leq r \qquad i=1,\dots,k.$$

Notice that now r is the unknown radius of the circumscribing circle that is minimized: its value depends on the set of circle radii $\{r_i\}$. Similarly to (5), model (6) has $2k+1$ decision variables, k convex nonlinear constraints, and $k(k-1)/2$ non-convex constraints.

To illustrate this model, in the last numerical example presented here we will pack circles of radius $r_i=i^{-1/2}$ $i=1,\dots,k$ into a circumscribing circle. Notice that in this case the total area of the embedded circles is slowly divergent as *k* goes to infinity: therefore the optimized radius also will be unbounded as a function of *k*. (Packings with bounded total area may also be of interest, of course.) Figure 3 shows the optimized circle arrangement found for k=20.

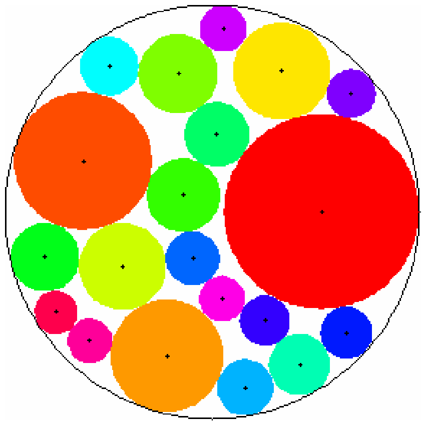

Figure 3. Packing 20 non-uniform size circles in the unit circle: $r_i=i^{-1/2}$ i=1,...,20

The radius of the circumscribing circle r=r(20) in this case approximately equals 2.12545. The corresponding runtime is about 47 seconds, on the machine mentioned before. Comparing this runtime with the previous one (that was 43 seconds for packing 20 identical size circles) one can see that MOP (i.e., LGO) handles the more general model with fairly little extra computational effort.

Although obviously all numerical test results depend also on certain solver parameterizations, we think that the examples presented indicate the capabilities and potentials of MOP. (The same default solver settings were used in all examples reviewed here, without any “tweaking”.)

Let us remark finally that we have attempted to solve a large variety of circle packing model instances using also other “general purpose” commercial optimization software products (and applying all solver options with default settings, the same way MOP was used). The solvers tested specifically included *Mathematica*’s built-in constrained optimization function (NMinimize), and several third party packages. Our comparative results consistently have demonstrated the relative strength and efficiency of MOP, both in terms of solution quality and runtime. These results will appear in a forthcoming paper, as well as in Kampas and Pintér (2005).

6. CONCLUSIONS

In addition to perhaps more "traditional" development environments – such as compiler platforms, spreadsheets, and algebraic modeling languages – integrated scientific-technical computing systems will play an increasing role in advanced systems modeling and optimization.

In order to meet related user demands, MathOptimizer Professional has been recently developed to handle nonlinear optimization problems formulated in *Mathematica*. MOP operations are based on an easy-to-use *Mathematica* interface to the LGO solver suite. Following a brief introduction to the key features of MathOptimizer Professional, we illustrate its usage by solving relatively small, yet non-trivial circle packing problems. More detailed numerical results and comparative assessments will appear elsewhere.

For over a decade, the core LGO solver suite has been applied in a large variety of research and professional contexts: consult, e.g., Pintér (1996, 2001, 2002, 2003, 2004, 2005), with numerous further references to such applications. In recent years, LGO has become a solver engine option available for use with an increasing number of modeling environments. Currently these include essentially "all" C and Fortran compilers, Excel spreadsheets, the GAMS modeling language, and the integrated scientific-technical computing systems Maple, *Mathematica* and MATLAB. (Further similar development is in progress.) The current LGO implementations have been used to solve models in up to a few thousand variables and constraints. We expect that MathOptimizer Professional will enable the solution of sizeable, sophisticated *Mathematica* models with efficiency comparable to that of compiler platform based nonlinear solvers.

ACKNOWLEDGEMENTS

The research of JDP related to *Mathematica* software development – including MathOptimizer Professional – has been supported by Wolfram Research, by providing quality software and professional documentation. We also wish to thank Mark Sofroniou of Wolfram Research for his permission to use (and to modify in our MOP development work) his Format.m package.

REFERENCES

Albers, D.J. (1996) A nice genius. *Math Horizons*, November 1996 issue, 18-23.

Aris, R. (1999) *Mathematical Modeling: A Chemical Engineer's Perspective*. Academic Press, San Diego, CA.

Bazaraa, M.S., Sherali, H.D. and Shetty, C.M. (1993) *Nonlinear Programming: Theory and Algorithms.* Wiley, New York.

Beltrami, E. (1993) *Mathematical Models in the Social and Biological Sciences*. Jones and Bartlett, Boston.

Benson, H.P. and Sun, E. (2000) LGO – versatile tool for global optimization. *ORMS Today 27* (5), 52-55. See also http://www.lionhrtpub.com/orms/orms-10-00/swr.html.

Bertsekas, D.P. (1999) *Nonlinear Programming.* (2nd Edition.). Athena Scientific, Cambridge, MA.

Bracken, J. and McCormick, G.P. (1968) *Selected Applications of Nonlinear Programming.* Wiley, New York.

Chong, E.K.P. and Zak, S.H. (2001) *An Introduction to Optimization*. (2nd Edition.) Wiley, New York.

Csendes, T. (2004) http://www.inf.u-szeged.hu/~csendes/publ.html.

Diwekar, U. (2003) *Introduction to Applied Optimization*. Kluwer Academic Publishers, Dordrecht.

Edgar, T.F., Himmelblau, D.M., and Lasdon, L.S. (2001) *Optimization of Chemical Processes.* (2nd Edition.) McGraw-Hill, NY.

Frontline Systems and Pintér Consulting Services (2001) *LGO Global Solver Engine for Excel – Premium Solver Platform.* Frontline Systems, Inc., Incline Village, NV. http://www.solver.com/xlslgoeng.htm.

GAMS Development Corporation and Pintér Consulting Services (2003) *GAMS/LGO User Guide.* GAMS Development Corporation, Washington, DC.
See http://www.gams.com/solvers/lgo.pdf

Gershenfeld, N.A. (1999) *The Nature of Mathematical Modeling.* Cambridge University Press, Cambridge, UK.

Grossmann, I., Ed. (1996) *Global Optimization in Engineering Design.* Kluwer Academic Publishers, Dordrecht.

Hansen, P.E. and Jørgensen, S.E., Eds. (1991) *Introduction to Environmental Management.* Elsevier, Amsterdam.

Horst, R. and Pardalos, P.M., Eds. (1995) *Handbook of Global Optimization, Vol. 1*. Kluwer Academic Publishers, Dordrecht.

Hillier, F. and Lieberman, G.J. (2005) *Introduction to Operations Research*. (8th Edition.) McGraw-Hill, New York.

Kampas, F.J. and Pintér, J.D. (2005) *Advanced Optimization – Scientific, Engineering, and Economic Applications with Mathematica Examples*. Elsevier, Amsterdam. (To appear.)

Maplesoft (2004a) *Maple*. Maplesoft, Inc., Waterloo, ON. http://www.maplesoft.com.

Maplesoft (2004b) *Global Optimization Toolbox.* Maplesoft, Inc. Waterloo, ON. http://www.maplesoft.com/products/toolboxes/globaloptimization/index.aspx.

MathWorks (2004) *MATLAB*. The MathWorks, Inc., Natick, MA.

Markót, M.Cs. (2004) http://www.inf.u-szeged.hu/~markot/publ.html.

Melissen, J.B.M. (1997) *Packing and Covering with Circles.* Ph.D. Dissertation, University of Utrecht.

Murray, J.D. (1983) *Mathematical Biology*. Springer-Verlag, Berlin.

Neumaier, A. (2004) Global Optimization. http://www.mat.univie.ac.at/~neum/glopt.html.

Operations Research: 50th Anniversary Issue (2002) INFORMS, Linthicum, MD.

Papalambros, P.Y. and Wilde, D.J. (2000) *Principles of Optimal Design.* Cambridge University Press, Cambridge, UK.

Pardalos, P.M. and Resende, M.G.C., Eds. (2002) *Handbook of Applied Optimization*. Oxford University Press, Oxford.

Pardalos, P.M. and Romeijn, H.E., Eds. (2002) *Handbook of Global Optimization, Vol. 2*. Kluwer Academic Publishers, Dordrecht.

Pardalos, P.M., Shalloway, D. and Xue, G. (1996) *Global Minimization of Nonconvex Energy Functions: Molecular Conformation and Protein Folding*. DIMACS Series, Vol. 23, American Mathematical Society, Providence, RI.

Parlar, M. (2000) *Interactive Operations Research with Maple: Models and Methods*. Birkhäuser, Boston.

Pearson, C.E. (1986) *Numerical Methods in Engineering and Science*. Van Nostrand Reinhold, New York.

Pintér, J.D. (1996) *Global Optimization in Action.* Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (2001) *Computational Global Optimization in Nonlinear Systems: An Interactive Tutorial.* Lionheart Publishing, Atlanta, GA.

Pintér, J.D. (2002) Global optimization: software, test problems, and applications, Chapter 15 (pp. 515-569) in: Pardalos and Romeijn, Eds. *Handbook of Global Optimization. Vol. 2*. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (2003) GAMS/LGO nonlinear solver suite: key features, usage, and numerical performance. (Submitted for publication.)
See also at http://www.gams.com/solvers/solvers.htm#LGO.

Pintér, J.D. (2004) *LGO – An Integrated Model Development and Solver Environment for Continuous Global Optimization. User Guide*. Pintér Consulting Services, Inc., Halifax, NS, Canada.

Pintér, J.D. (2005) *Applied Nonlinear Optimization in Modeling Environments.* CRC Press, Boca Raton, FL. (To appear.)

Pintér, J.D. and Kampas, F.J. (2003) *MathOptimizer Professiona.User Guide*. Pintér Consulting Services, Inc., Halifax, NS, Canada.

Rich, L.G. (1973) *Environmental Systems Engineering.* McGraw-Hill, Tokyo.

Salford Software (2004) *Salford Fortran 95*. http://www.salfordsoftware.co.uk/

Schittkowski, K. (2002) *Numerical Data Fitting in Dynamical Systems*. Kluwer Academic Publishers, Boston / Dordrecht / London.

Specht, E. (2004) www.packomania.com.

Tawarmalani, M. and Sahinidis, N.V. (2002) *Convexification and Global Optimization in Continuous and Mixed-integer Nonlinear Programming.* Kluwer Academic Publishers, Dordrecht.

TOMLAB Optimization Inc. and Pintér Consulting Services (2004) *TOMLAB /LGO User Guide.* http://tomlab.biz/docs/TOMLAB_LGO.pdf.

Wilson, H.B., Turcotte, L.H., and Halpern, D. (2003) *Advanced Mathematics and Mechanics Applications Using MATLAB*. Chapman & Hall / CRC Press, Boca Raton, FL.

Wolfram Research (2004) *Mathematica.* Wolfram Research, Inc., Champaign, IL. http://www.wolfram.com.

Zabinsky, Z.B. (2003) *Stochastic Adaptive Search for Global Optimization.* Kluwer Academic Publishers, Dordrecht.

Zwillinger, D. (1989) *Handbook of Differential Equations*. (3rd Edn.) Academic Press, New York.