RECLAMATION

Managing Water in the West

Technical Report S&T-2011-486

Advanced Algorithms for Hydropower Optimization

U.S. Department of the Interior
Bureau of Reclamation
Technical Service Center
Denver, Colorado

March 2012

Legal Notice

Cover photo: A view of Glen Canyon Dam jet tubes during the 1996 Beach/Habitat Building Flow (Reclamation file photo).

Technical Report S&T- 2011- 486

Advanced Algorithms for Hydropower Optimization

This Research Report was prepared by the U.S. Department of the Interior, Bureau of Reclamation, Technical Service Center, Denver, Colorado.

David A. Harpman 2/22/2012

Prepared by: David A. Harpman, Ph.D. Date
Natural Resource Economist
Economics and Resource Planning Team, 86-68212

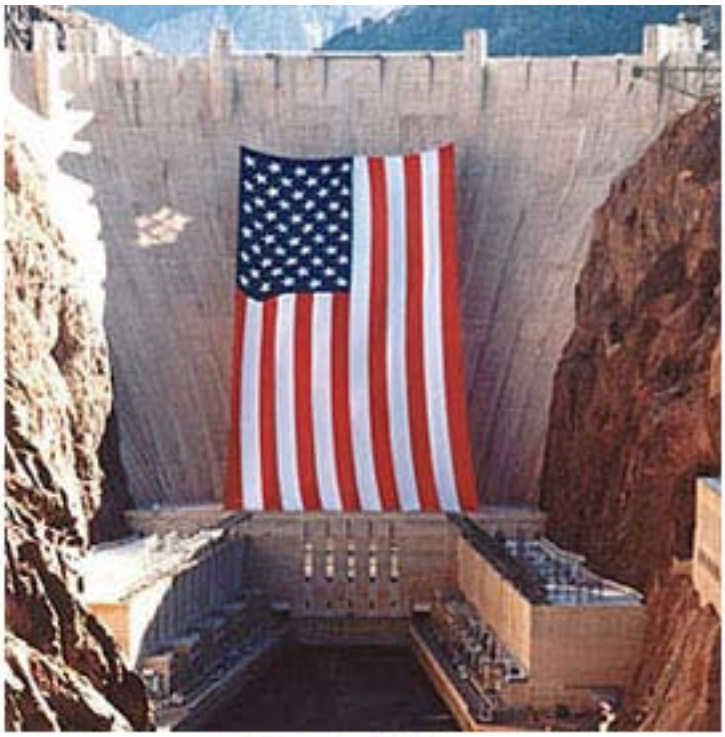
2/24/2012

Peer Review: Jonathan L. Platt, MSc. Date
Natural Resource Economist
Economics and Resource Planning Team, 86-68212

2/27/2012

Peer Review: Steven M. Coberly, P.E. Date
Electrical Engineer
Hydropower Technical Services, 86-68440

REVISIONS					
Date	Description	Prepared	Checked	Administrative Approval	Peer Review



Hoover Dam

Celebration of Reclamation's 100th Anniversary at Hoover Dam.

Acknowledgments

This document has benefited immeasurably from the gracious assistance and technical guidance provided by the collaborative research team. In alphabetical order, the members of the team are:

Dr. Craig A. Bond, Colorado State University
Dr. Darrell G. Fontane, Colorado State University
Dr. George W. Heine, U.S. Bureau of Land Management
Mr. Thomas D. Veselka, Argonne National Laboratory

Any errors are the sole responsibility of the author.

Project Funding

This research project was funded in Fiscal Years 2010 and 2011 by the U.S. Bureau of Reclamation's Science and Technology Program, Project Identification Number 486. Additional support from the University of Denver, Colorado State University and Argonne National Laboratory is gratefully acknowledged.

Credits

Dr. David Raff, Hydraulic Engineer, U.S. Bureau of Reclamation, Denver, Colorado, constructed the graphics shown in Appendix 14 using MATLAB version 4.x. His technical assistance is gratefully acknowledged.

Mr. Thomas Veselka, Energy Systems Engineer, Argonne National Laboratory, contributed substantive portions of the narrative and graphics in Appendix 2. Figure 18 is reproduced from Veselka et al. (2010b) and used with the permission of the ever-gracious Mr. Veselka.

Contents

Page

Tables

Page

Figures

Page

Executive Summary

The advent of personal computers in the mid-1980s gave rise to an era of unparalleled advances in heuristic optimization research. These new optimization algorithms are not based on traditional calculus-based approaches, but instead have their origins in physical and biological processes. Three promising evolutionary algorithms (EAs) were identified from the emerging literature; the real coded genetic algorithm (RCGA), differential evolution (DE) and particle swarm optimization (PSO). These EAs were then applied to an important hydropower problem—the constrained dynamic economic dispatch problem. A suite of replicated experiments were conducted to assess their performance characteristics. These experiments were used to compare the performance of these EA's with a traditional solution approach, and to explore the influence of initialization approaches, convergence criteria, the dimensions of the problem, the role of problem inputs and the effects of binding constraints. Relative to traditional calculus based approaches, these three evolutionary algorithms exhibit longer solution times—characterized by rapid identification of the region containing the optimum, followed by a slow convergence on the optimum. For this problem, the choice of initialization approach appears to have no appreciable effect on solution times. Convergence times become longer as the problem size increases and, for some of the algorithms, when constraints are binding. The reliability of the EA's proved to be excellent and their convergence speeds are acceptable for use in operational decision-making. As a practical matter, the majority of applied hydropower optimization problems are non-convex and discontinuous. These conditions preclude the use of traditional calculus based algorithms. In contrast, evolutionary algorithms are robust under these conditions. Continuing development and testing of these algorithms, leading towards operational deployment, is now ongoing.

Introduction

Within the last 30 years, a variety of new optimization heuristics have been described in the power engineering literature. These heuristic approaches rely on innovative search techniques, drawn from biological and physical processes. Although computationally intensive, these methods can solve difficult constrained optimization problems, like the dynamic economic dispatch problem, quickly and reliably.

This research describes several of the most promising of these new optimization approaches, applies them to example economic dispatch problems and systematically assesses their performance. The goal of this effort is to identify

algorithms which can help guide the hydropower economic dispatch decisions and improve efficiency, generating more electric power with less water.

Reclamation plays a highly visible leadership role in the electric power industry. The Bureau is the second largest producer of hydroelectric power in the United States and, if it were a utility, would rank as the ninth largest electric utility on the basis of production capacity. We operate 58 power plants with an installed capacity of 14,809 MW and produce approximately 40 billion kWh of energy annually. As a registered generation and transmission owner-operator, Reclamation plays a key role in regional reserve sharing agreements and is a member of the Western Electricity Coordinating Council (WECC).

By statute, Reclamation's electric power must be marketed at the lowest possible rates consistent with sound business practice. The goal of this research project is to identify and apply advanced approaches allowing the operation of Reclamation hydropower plants in a more efficient manner generating more electricity per acre-foot of water released. This research is fully consistent with Reclamation's mandate and reflects our stewardship responsibilities as a water and power provider. Improved efficiency will result in the generation of more electric power using less water benefitting water and power users, as well as the American taxpayer.

Hydropower Problems

The focus of this research effort is on the dynamic economic dispatch problem which is of particular importance to Reclamation and other hydropower owner/operators. There are several mathematically related hydropower problems which can sometimes prove confusing. To ensure a common basis for understanding, three of these problems are summarized in this section. These are the (static) economic dispatch problem, the dynamic economic dispatch problem and the unit commitment problem.

Economic Dispatch

The static economic dispatch problem is a mathematical optimization problem which identifies how to optimally manage one or more operating hydropower units together in a single time-step, typically a one hour period. This problem assumes the hydropower units have been previously committed or operating. Static economic dispatch problems typically consider the minimum and maximum output constraints of each available unit as well as their engineering characteristics, such as head, release and efficiency characteristics.

Dynamic Economic Dispatch

The dynamic economic dispatch problem, which is the focus of this research, is an extension of the static dispatch problem. It is a mathematical optimization problem which can identify how to optimally manage one or more hydropower units over a specified time horizon. The time horizon considered might consist of a day (24-hours), a week (168-hours) or some other period. Like the static problem, the dynamic economic dispatch problem also assumes the hydropower units are currently committed or operating. Typically such problems consider the minimum and maximum output constraints of each available unit, and their engineering characteristics. A unique feature of dynamic economic dispatch problem is their consideration of a multiple time-step planning horizon and their ability to include time-step to time-step ramp rate constraints on each unit, or combination of units.

The dynamic economic dispatch problem underlies many high visibility hydropower planning analyses. Its efficient and accurate solution is of paramount importance in such studies. Prominent Reclamation examples include studies of the economic impacts of changing operations at the Aspinall Unit (Veselka et al 2003) and Glen Canyon Dam (Harpman 1999).

Unit Dispatch

The unit commitment or unit dispatch problem is a complex 2-step mathematical optimization problem. Solution of this problem requires (a) identification of the combination of available hydropower units to operate (or shut down) in a single time-step, such as a 1–hour period, and, (b) how to managed the committed hydropower units in an optimal manner. The decision to operate or shut down a unit is a binary (0/1) decision, typically with some associated cost and often with some minimum time constraint imposed between startup and shutdown decisions. Assuming there are n available hydropower units, the unit dispatch problem is of size 2^n. This aspect of the problem poses a potentially large and difficult-to-solve integer programming effort. Once the optimal units have been committed, the economic dispatch problem, described previously, is solved for those units.

The Dynamic Economic Dispatch Problem

The hydropower plant operator is faced with a challenging dynamic optimization problem. Given the amount of water available for release and the anticipated price of electricity over a particular time horizon (T), the plant operator must decide how much water to release for generation in each period (t) in order to maximize the economic value of the electricity produced. Typically, the total

amount of water available for release (Q) over the planning horizon is fixed and known. The vector of prices (R) over the planning horizon (T) is assumed or anticipated, based on prior experience and knowledge.

In general, the optimal dynamic dispatch problem can be written in mathematical notation as shown in equations (1) through (6).

(1) $$\textit{Maximize} \quad \sum_{1}^{T} R_t \, p_t(q_t)$$

subject to:

(2) $$\sum_{1}^{T} q_t \le Q$$

(3) $$p_t \ge 0$$

(4) $$q_{\min} \le q_t \le q_{\max}$$

(5) $$p_{\min} \le p_t \le p_{\max}$$

(6) $$abs(\Delta q) \le rrate$$

Where:
R_t = price ($/MWh) at time (t)
p_t = generation (MW) at time (t)
q_t = release (cfs or af) at time (t)
Q = total release (af).
q_{min} = minimum release.
q_{max} = maximum release
p_{min} = minimum generation level.
p_{max} = maximum generation level
Δq = change in q from t to t+1.
rrate = maximum ramp rate.

In practice, the operator attempts to maximize economic value over the time horizon by producing electricity when it is most valuable. While doing so, s/he cannot exceed the amount of water available for release over the time horizon (equation 2), must respect the minimum and maximum release levels (equation 4), must respect the minimum and maximum generation levels (equation 5) and must necessarily accommodate the ramp rate limitations of the plant in moving from one release level to the next, if any (equation 6).

This problem falls into the class of mathematical problems known as constrained optimization problems. Depending on the nature of the generation and head relationships, the problem may be highly nonlinear.

Solution of the optimal dynamic dispatch problem represents a daunting mathematical undertaking. In general, it is not possible to solve this type of problem analytically and highly specialized computer software must be employed to obtain a numerical solution to the problem. The efficient and rapid solution of this class of problem is the focus of this research effort.

The dynamic economic dispatch problem, as it has been described here, reflects a purely economic objective. This approach is employed in the vast majority of economic, engineering and research applications. In some settings, the operator's motivations are more complex and their objectives may vary from this purely economic approach. Appendix 1 describes several different but plausible objective functions. Use of any of these alternative objective functions, will lead to optimal solutions which may differ in character from those obtained in this study.

The dynamic economic dispatch objective used in this study is a simplified version of reality since it does not allow for the possible provision of ancillary services by the hydropower plant. Ancillary services are electrical products, other than energy generation, which help maintain reliable system operations in accordance with good utility practice. Two ancillary services in particular, spinning reserves and regulation, can affect the optimal economic dispatch decision. Appendix 2 further discusses ancillary services and describes how their provision might be expected to affect economic dispatch.

Appendix 5 sketches out the analytic solution of the dynamic economic dispatch problem using calculus. This appendix is particularly useful because it contains a specific example which draws upon the details of the problem explored throughout the remainder of this document.

Optimization Review

Purpose

The focus of this report is on the optimization of functions which are commonly encountered in hydropower operations. This section serves as a review of this relatively specialized topic. It provides some relevant background including a common understanding of the mathematical terms used throughout the remainder of this document, as well as some high-level insights into the mathematics of optimization.

Stationary Points

A stationary point is a point for which the first derivative of the function is zero. A stationary point can be a point of inflection, a saddle point or an extrema. A point of inflection is a point where the derivative changes sign from positive to negative. A saddle point is a point in n-dimensions for which the first derivatives are zero, but which is not an optima. Such points are caused by the coincidence of say, a maximum point in one or more dimensions and a minimum point in one or more different dimensions. An extremal point is a "true" optima, which can be either a minimum or maximum point.

Extremal Points

Over the domain of permissible real number values, a function may have a variety of so called, "extremal points." In the realm of mathematical optimization, these are known as extremum, extrema, minima, maxima, minimum points or maximum points, depending on the author and the context. Alternatively, these same points may be called, "optima" or optimal points. Again, the choice of terminology varies with the author's style and discipline.

Global and Local Extremal Points

A generic nonlinear function is plotted in two dimensions (2D) over the real number domain [-∞, +∞] in Figure 1. This generic nonlinear function is multimodal. In other words, it has multiple extremal points, a characteristic which is not uncommon in applied work. Multiple local extrema are known by various terms including multiple local optima, multiple optima and/or multiple local optimal points. Some authors refer to such functions as ill-behaved or complex.

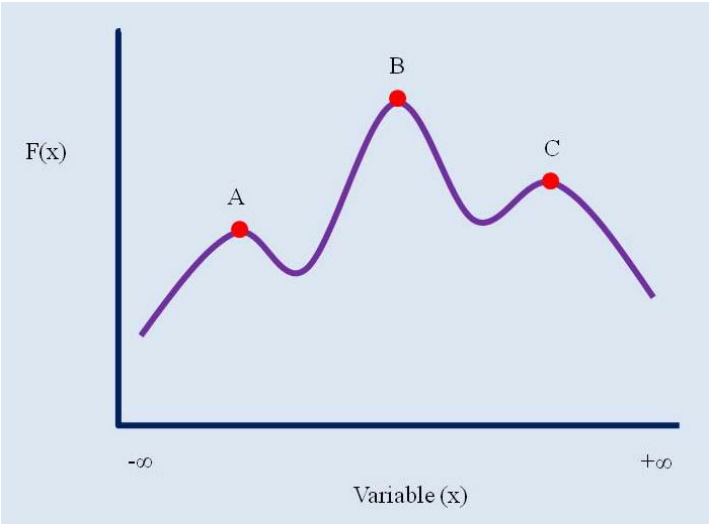


Figure 1. Nonlinear function with multiple local extrema.

The generic nonlinear function shown in Figure 1 has a single global extrema or optima, in this case a maximum, at point B. The function also has two local optima, one at point A and the other at point C.

Identification of the global optima (point B) for a nonlinear function such as this one can be extremely difficult for traditional calculus based optimization algorithms. Typically, these algorithms will identify the extremal point which is closest to their given starting position. For example, if the optimizer is started at a point to the right of point C, it will generally converge on point C. Point C is the local, rather than the global optima.

Convex and Concave Functions

In general, it is not possible to prove mathematically that a given local extrema is the global extrema, except in the case of mathematical functions with certain specific characteristics known as convex or concave functions. These terms have very specific mathematical definitions (see Boyd and Vandenberghe (2006) for the many details). The general concepts are readily illustrated in two dimensions (2D), avoiding a lengthy theoretical discussion. A function which is (strictly) convex (from above) is shown in Figure 2 panel A. As shown in this figure, a line drawn between any two points on a convex function will not intersect the function at any other point. A function which is (strictly) concave (from above) is shown in Figure 2 panel B. As shown in this figure, a line drawn between any two points on a concave function will not intersect the function at any other point.

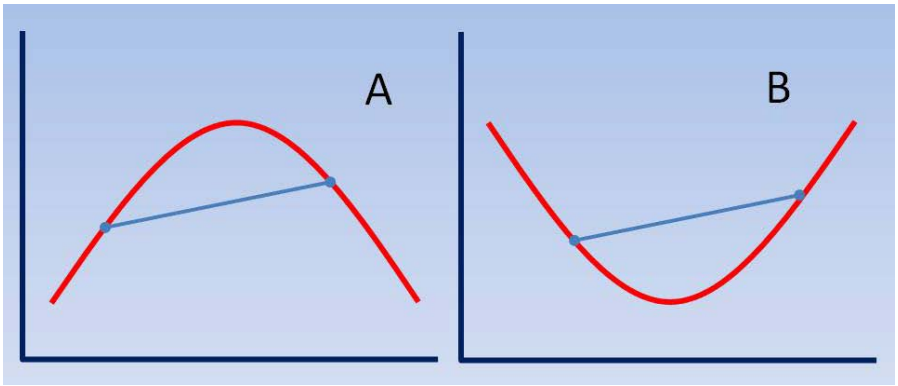


Figure 2. Convex (panel A) and concave (panel B) functions.

Convex and concave functions have appealing properties which facilitate mathematical optimization. Strictly convex functions, such as the example shown in Figure 2 panel A, have a single (unique) maxima, and that single extrema is the global maximum. Strictly concave functions, such as the example shown in Figure 2 panel B, have a single (unique) minima, and that extrema is the global minimum point.

Calculus of Unconstrained Optimization

Since the time of Sir Isaac Newton (circa 1400), mathematicians, economists and engineers have collectively devoted vast amounts of effort to the study of optimization, with a particular focus on convex optimization problems with linear constraints. Not surprisingly, calculus based optimization approaches are routinely taught to all engineers and economists. Most students of these disciplines have wonderful memories of their many calculus courses.

Although exposure to these topics is rather wide-spread, a cursory review of calculus based optimization is a useful digression. All students of calculus will recall the first order necessary conditions (FOC's) for the existence of an optima are that the first derivatives are equal to zero. This is illustrated mathematically in equation (7).

$$\frac{\partial f(x_i)}{\partial x_i} = 0 \tag{7}$$

An alternative but equivalent restatement of this condition is that the gradient, or vector of first partial derivatives, is equal to zero (equation 8).

$$\nabla f(x) = 0 \tag{8}$$

Points where this occurs are points where a line tangent to the function is horizontal, or flat. This occurs at all stationary points which may be a point of inflection, a saddle point or an optimal point. While all optima are flat spots, not all flat spots are optima. At a saddle point, for example, the vector of first derivative is zero, but it is not an extrema. Consequently, the first order conditions are said to be a necessary, but not sufficient condition for identifying an optimum point.

The sufficient or second order sufficient condition (SOC) for indentifying an optima is that the Hessian matrix, or matrix of second partial derivatives, must be negative definite (for a maxima) or positive definite (for a minima). A matrix is said to be positive definite if $z^T Mz > 0$ for all non-zero vectors z with real entries. A matrix is said to be negative definite if $z^T Mz < 0$ for all non-zero vectors z with real entries. The determinate of a real symmetric positive definite matrix is strictly positive (all eigenvalues are positive). The determinate of a real symmetric negative definite matrix is strictly negative (all eigenvalues are negative).

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} < 0 \text{ for a maximum} \tag{9}$$

(10) $$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} > 0 \text{ for a minimum}$$

The second order conditions are used as a test to ensure the vector of derivatives is not only zero everywhere, but is diminishing in all dimensions (for a maximum) or increasing in all directions (for a minimum). If the first order necessary conditions and the second order conditions hold at a particular point, we can be sure the identified point is at least a local optimal point.

Limits of Calculus Based Optimization

Calculus based optimization is elegant and efficient but has some practical limitations. Application of calculus based optimization techniques is limited to functions which are smooth, continuous and twice-differentiable. Calculus based optimization technologies cannot (readily) be applied to cases where derivatives either do not exist or cannot be calculated. Objective functions which are discontinuous or are undefined arise relatively frequently in hydropower optimization problems.

Selected Terms

Like any branch of science, there are some terms used to describe mathematical optimization approaches which are not commonly encountered in other fields. As an aid to understanding the narrative which follows, it will be useful to define some of these terms.

Algorithm

"A detailed sequence of actions to perform to accomplish some task. Named after an Iranian mathematician, Al-Khawarizmi. Technically, an algorithm must reach a result after a finite number of steps, thus ruling out brute force search methods for certain problems, though some might claim that brute force search was also a valid (generic) algorithm. The term is also used loosely for any sequence of actions (which may or may not terminate)" (Computer Dictionary Online 2010).

Heuristic

"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Unlike (true) algorithms, heuristics do not guarantee optimal, or even feasible, solutions

and are often used with no theoretical guarantee" (Computer Dictionary Online 2010).

In practice, the term algorithm is often used interchangeably with the term heuristic. However, mathematicians typically reserve their use of the word algorithm to describing optimization approaches for which there is a theoretical mathematical basis for expecting a favorable result. Typically, mathematicians employ the term heuristic to describe any of the non-traditional optimization approaches not supported by mathematical theory.

Objective Function

The object of mathematical optimization is to minimize or maximize a specified mathematical expression. This expression is known as an objective function.

Penalty

Many applied mathematical optimization problems have natural or logical constraints on the values which can be considered in the solution. For example, physical (quantity) measurements are typically non-negative.

One approach to characterizing constraints in a constrained mathematical optimization problem is to arithmetically disadvantage, or penalize, solution results which violate a constraint. This topic is discussed in much greater detail in subsequent sections of this document. A penalty function is used to compute the numerical magnitude of the disadvantage caused by one or more constraint violations. A penalty is the value returned by a penalty function.

Fitness

In cases where penalty functions are used to characterize constraint violations, a fitness function is maximized or minimized instead of an objective function. A fitness function returns the numerical value of the fitness—defined as the objective function value plus the value of the penalties for constraint violations, if any.

Optimization Approaches

Taxonomy of Optimization Approaches

For purposes of this document and the discussion which follows, it will prove useful to provide some type of taxonomy or classification scheme to illustrate the relationships between these two optimization approaches. Figure 3 provides some structure for this discussion.

As shown in Figure 3, optimization approaches can be divided into traditional (calculus based) optimization algorithms and heuristic algorithms. The latter class of optimization methods may also be described as metaheuristics or heuristic optimizers, depending on the author and the source.

The focus of this research is on a sub-set of optimization methods which are classified as heuristic algorithms. Even so, comparison and understanding of these methods is facilitated by some familiarity with traditional methods and approaches.

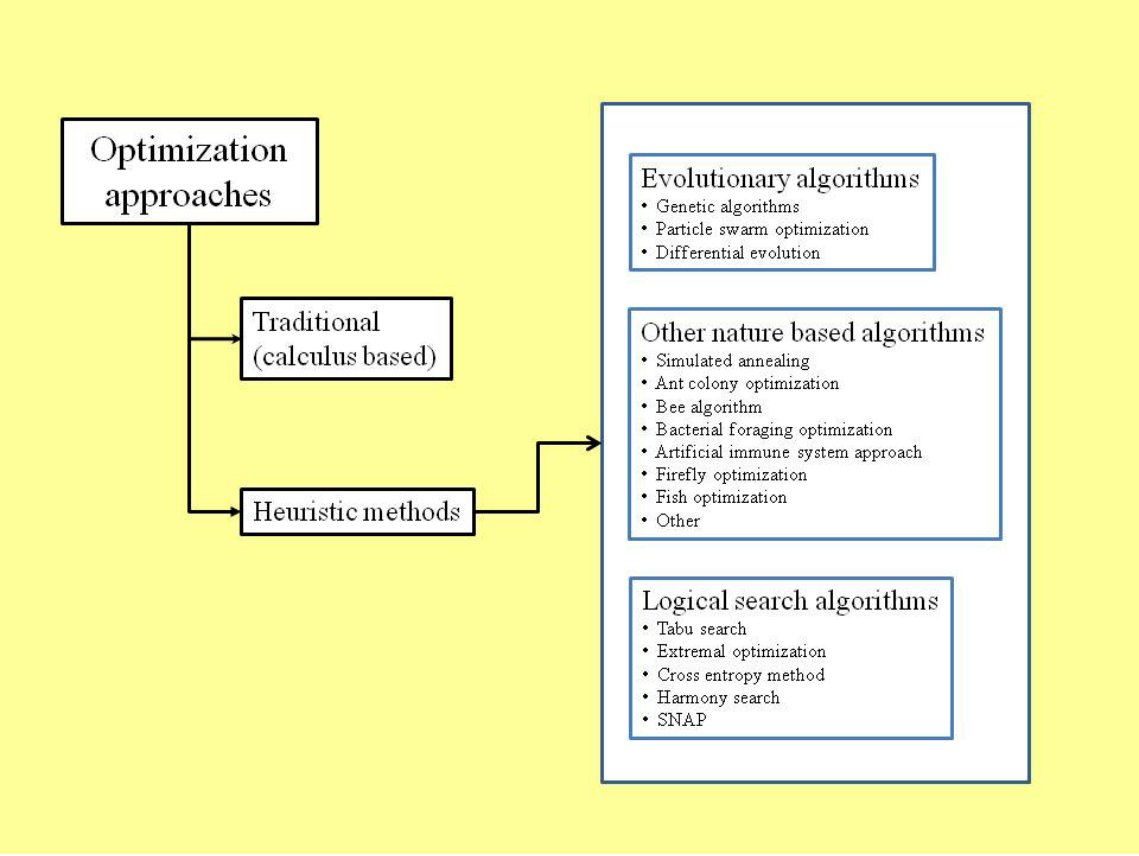


Figure 3. Taxonomy of optimization approaches.

Traditional Solution Algorithms

Optimization problems have traditionally been addressed with a variety of traditional calculus based methods and throughout the remainder of this document, these approaches will be referred to as "traditional" or calculus based approaches. Calculus based optimization approaches are routinely taught to all engineers and economists. Most students of these disciplines will surely have fond memories of the many hours they devoted to mastery of this topic!

Since the time of Sir Isaac Newton (circa 1400), mathematicians, economists and engineers have collectively devoted vast amounts of effort to the study of optimization, with a particular focus on convex optimization problems with constraints. There are many books devoted to this subject, one of the many modern examples being the tome by Boyd and Vandenberghe (2006).

Numerical solution of convex optimization problems is typified by the Newton-Raphson approach and its many variants. This approach has been taught to engineers and economists since the early 1950's (for example, see Wood and Wollenberg (1996) or Rau (2003)) and is the subject of Appendix 6 in this document.

As described in Press et al (1989) and Judd (1999), the Newton-Raphson approach has been largely supplanted by some of its recent and more advanced variants. At the present time, two approaches are in the forefront of current calculus based optimization technology. These are the sequential quadratic programming (SQQ) method, and, the generalized reduced gradient (GRG) method. Both of these methods are aptly described in Rau (2003). The SQQ method is often used in high-end commercially available optimization platforms, such as LINGO (www.lindo.com). The GRG method has found its niche as the optimization solver incorporated in all currently shipping versions of Microsoft Excel (Fylstra et al 1998). As such, it may well be the world's most frequently used optimization algorithm. In any case, it is almost certainly the most widely installed optimization package! As bundled with the ubiquitously available Excel program, the solver is broadly employed in graduate and undergraduate teaching (for example, see Weber 2007).

Heuristic Optimization Methods

The focus of this research is on the application of a subset of the heuristic optimization methods shown in Figure 3. Heuristic optimization approaches are based on the application of rules and logic which reduce the search space and allow for solution of difficult optimization problems. Generalizing rather broadly, we can classify these methods into the three categories shown; evolutionary algorithms, other nature based algorithms and logical algorithms.

Evolutionary algorithms explicitly characterize crossover, mutation and selection operators (Engelbrecht 2005). As might be expected by their name, evolutionary algorithms are based on the concept of biological evolution. These approaches are based on the improvement of an artificial population of individuals over a series of generations or iterations. Each individual carries a solution to the optimization problem. At each generation, the most fit individuals in the population reproduce and their offspring survive into the next generation, the less fit individuals die and their inferior genes are lost. The fitness of the population and the quality of the solutions found, improve over time. Genetic algorithms, differential evolution and particle swarm optimization fall into this category of algorithms.

There are an amazing variety of optimization heuristics related to living organisms, their behavior or some other natural physical phenomenon. Among these are ant colony optimization, bee optimization, firefly optimization and a host of others. As might be surmised, some of these algorithms are predicated on the collective food location strategies typified by the species.

For purposes of this document, we will classify these remaining approaches as logical heuristic search algorithms. While these may be very different from one another in search strategy, they are based on logical insights, experience and in-depth knowledge of one or more types of optimization problems. As shown in Figure 3, this category includes such well-known heuristics as Tabu search and Extremal optimization. It also includes some less well known but quite effective algorithms such as the Substitution-based Non-linear Approximation Procedure (SNAP) algorithm developed by Veselka, Schoepfle and Mahalik (2003)

Comparison of Approaches

Much of the research effort described in this report is focused on the application of evolutionary algorithms to a common hydropower optimization problem. A comparison of these two classes of algorithms and their respective suitability to this problem will provide both some background and rationale. Table 1 compares a number of pertinent characteristics of these two types of approaches.

The hydropower problems examined here are inherently nonlinear with both nonlinear and linear constraints. Both traditional and evolutionary algorithms can be applied to these types of problems. Very fast and incredibly reliable traditional algorithms are available for solving problems with linear objective functions and constraints. However, traditional algorithms are typically less efficient when applied to nonlinear objectives and nonlinear constraints. They typically require longer solution times and can fail to identify a solution more frequently in this setting.

Table 1. Traditional and Evolutionary Algorithms

Characteristic	Traditional Algorithms	Evolutionary Algorithms
Problem formulation	Linear or nonlinear	Linear or nonlinear
Mathematical requirements	Smooth, continuous and twice differentiable	Can be piecewise, discontinuous and non-differentiable
Allowable constraints	Equality, inequality, linear or non-linear.	Equality, inequality, linear or nonlinear.
Mathematical requirements	Calculus, linear and matrix algebra operations	Primitive mathematical operators only (add, subtract, multiply, divide)
Function return	Single solution	Multiple solutions
Nature of outcome	Deterministic	Stochastic
Optimal point	Extremal point closest to starting position usually identified. This may or may not be the global optima.	Extremal point within search range usually identified. This is more likely to be the global optima.
Memory requirements	Extensive	Modest
Convergence characteristics	Slow large-scale search Fast local convergence	Fast large-scale search Slow local convergence
Solution time	Short	Often lengthy
Code implementation	Complex (very)	Unsophisticated

Many commonly encountered hydropower problems are nonlinear, nonconvex, and have discontinuities. This includes the dynamic economic dispatch problem and the unit dispatch problem examined here. Perhaps the chief strength of evolutionary programs is their applicability to these types of real-world hydropower problems, a factor which largely motivated this research effort. The mathematical requirements for applying traditional optimization algorithms are rather restrictive. Typically, traditional algorithms can only be employed when the objective function and the constraints are smooth, continuous and twice differentiable. In contrast, evolutionary algorithms can solve a much wider range of problems including those which are discontinuous, piecewise, are not convex and which cannot be differentiated.

Both traditional and evolutionary algorithms can solve constrained optimization problems with various types of constraints including equality, inequality, linear and nonlinear constraints. Traditional algorithms are less well suited to solving optimization problems with nonlinear constraints. The solution of problems with one or more equality constraints can be problematic for evolutionary algorithms.

The mathematical requirements for implementing evolutionary algorithms are far less onerous than they are for traditional (calculus based) algorithms. In both philosophy and practice evolutionary algorithms are not based on calculus and do

not use calculus constructions for obtaining a solution. In fact, some authors consider this to be their greatest strength! Evolutionary algorithms use only primitive mathematical operators such as addition, subtraction, multiplication and division. Traditional algorithms are, of course, founded in calculus concepts. As a result, they use not only gradient vectors (vectors of first partial derivatives) and hessian matrices (matrices of second partial derivatives), but also have advanced linear algebra requirements. These advanced mathematical constructs are error prone to derive and code, difficult to implement numerically and require an extremely high degree of knowledge and skill on the part of the researcher/programmer. Judd, a master of understatement, writes "Many readers could write acceptable unconstrained optimization code, but it is much more difficult to write good, stable, reliable code for constrained optimization (Judd 1999, page 142)

Traditional (calculus based) optimization algorithms return one single solution. It is *the* solution to the problem, as every economics and engineering student is acutely aware. A fundamental difference between traditional and evolutionary algorithms is that evolutionary algorithms return a population of solutions. This difference in solution paradigm is both unfamiliar and potentially confusing.

To expand upon this concept, we must recall that evolutionary algorithms characterize a population of individuals. This population is of say size, np, which could consist of from 5 to 100 individuals or more. Fundamentally, each of these np individuals stores a solution (in some cases, more than one). The stored solution consists of not only the optimal function value, but the vector of values which produces it. As the evolutionary process proceeds, each of these np solutions evolves and becomes better, or more "fit." When the evolutionary process terminates, the result is np, not necessarily unique, individual solutions--not one single solution. As a practical matter, the analyst will often choose to report the best of these np individual solutions as *the* solution. Since evolutionary algorithms are probabilistic in nature, each new run will produce slightly different results (in contrast with a traditional algorithm which produces identically the same result for a given starting condition). In the case of evolutionary algorithms, it is customary to undertake multiple runs and report the mean and other descriptive statistics for the outcomes.

Many real-world optimization problems have more than one optimal or extremal point. At an extremum, the first order necessary conditions (FOCs) for a minimum or maximum are satisfied. In the case of a traditional calculus based algorithm, the specific extrema identified by the algorithm depends primarily on the starting conditions specified by the analyst. These types of functions are the bane of researchers everywhere! In the absence of detailed knowledge about the optimal surface, the usual procedure is to restart the traditional algorithm at many different points in the solution space and search for the global optimum point. Problems which exhibit multiple local optima can often be solved by these calculus based methods. However, there is no theoretical or practical way to

guarantee the solution identified by the researcher is the global solution to the problem.

Evolutionary algorithms are sometimes described as global optimizers owing to their well-documented ability to identify the global optima within the given search space. Notwithstanding the published glowing reports, an equal body of published evidence suggests this behavior is not universally observed. Furthermore, it cannot be proved theoretically that they can be relied upon to identify the global best solution. It is most certainly true that relative to traditional algorithms, evolutionary programs carry more solutions through the iteration process and have much greater exploratory ability. These two characteristics enable evolutionary algorithms to more exhaustively traverse the solution space. Consequently, they are much more likely than traditional algorithms to identify the global optima.

Traditional optimization algorithms make heavy use of vectors, matrices and linear algebra operations, which themselves exact a huge computer memory overhead. Consequently, traditional optimization algorithms require extensive amounts of computer memory, especially for the solution of sizable problems. As little as ten years ago the practical usage of traditional optimization algorithms was restricted by the amount of physical and virtual memory addressable by existing microcomputers. In contrast, evolutionary algorithms do not make use of vectors, matrices or other advanced mathematical structures or operators. Their memory requirements are quite modest for similar size problems.

In cases where they can be applied, traditional calculus based optimization algorithms are known for their rapid converge properties. This is especially true in the case of convex functions with linear constraints. Experiments show that for traditional optimization algorithms, the initial phases of search are quite slow. Once they have identified the region where the optima resides, local convergence to the final solution is often very fast. Evolutionary algorithms on the other hand, exhibit behavior which is very much the opposite. Experiments on evolutionary algorithms demonstrate the initial search phase is very fast—the algorithms quickly and efficiently locate the region of the optima. However, the local convergence of these algorithms is slow, in some cases, painfully so. Typically, large amounts of time are required for the population to converge on an optimal point, after the region where it is located has been isolated.

The computational resources required by traditional calculus based algorithms and evolutionary algorithms differ profoundly. Not surprisingly, the time required to achieve convergence is vastly different. Traditional algorithms require large amounts of memory but typically require less than 100 major iterations to converge to a solution. Evolutionary algorithms often require thousands or tens of thousands of iterations to converge to a solution. While it is true that evolutionary algorithms utilize only primitive mathematical operations—it is no understatement to say they do so intensively! Prior to the advent of

microcomputers, the lack of sufficient computing power and sheer cost of computer resources precluded the use of evolutionary algorithms for civilian purposes.

One of the advantages of evolutionary algorithms is their ease of implementation. Unlike traditional algorithms, effective cutting-edge evolutionary algorithms are routinely developed by researchers and hobbyists. As of December 2010, there a number of toolboxes and working computer codes are available. Even so, many researchers with limited resources, develop research grade evolutionary algorithms using high level computer languages such as C++, C, Fortran, Java, Visual Basic and Delphi. This is rarely the case for traditional calculus based algorithms.

Heuristics and Microcomputers

Heuristic algorithms are computationally intensive and scientific advances in heuristics are necessarily related to the nearly unimaginable innovations in computer technology made within the last thirty years. Arguably, there are two fundamental aspects of this evolution—vast improvements in computational speed, and, the widespread availability of microcomputers.

The computational resources required by traditional calculus based algorithms of yesteryear and modern evolutionary algorithms differ profoundly. Traditional algorithms require relatively large amounts of available memory but typically require less than 100 major iterations to converge to a solution. In contrast, evolutionary algorithms often require thousands, tens of thousands or even millions of iterations to converge on a solution. While evolutionary algorithms utilize only primitive mathematical operations—they do so intensively!

Although this fact is often overlooked by the young, computers are a relatively recent invention. Depending on the source, the first fully programmable computer was debuted in the 1940s. These early computers were large centralized hardware installations which we now describe as, “mainframe” computer architectures. Relative to the current norms, they were incredibly expensive, slow and ponderous. Access to the then existing computational resources was rationed and limited to a few elite civilian researchers, and members of the defense establishment. Experiments using unproven technologies or computationally intensive processes were exceedingly rare.

The advent of microcomputers changed this paradigm. The Apple II personal computer was introduced in 1977 and the International Business Machines (IBM) Company marketed their first computer in 1981. Microcomputers were designed to be used independently of institutional controls and shared usage constraints. They could be purchased relatively cheaply by individual researchers, and perhaps most importantly-- were consistently and conveniently available for use.

Even the early microcomputers were technically and numerically capable tools. As further technological innovations were made, hardware costs (memory and storage) fell dramatically and computational speeds increased. These characteristics made it possible for established mainstream researchers, as well as hobbyists and researchers working at the fringes of established theory and practice, to purchase microcomputers and to experiment with their ideas freely and at little cost.

Modern researchers often take for granted the massive computational power at their disposal. Since this is particularly true in the case of younger researchers, a brief divergence will provide a useful point of reference. The pace of hardware and speed improvements since the appearance of personal computers has been dizzying. For example, the Apollo 11 mission in 1969, which successfully landed men on the moon, used an onboard computer which had eight times less memory and ran at a much lower speed than the IBM XT personal computer released in 1981 (Robertson 2009). The basic configuration of a 1981 IBM XT computer had 16 kilobytes (0.000016 gigabytes) of random access memory (RAM), 10 megabytes (0.010 gigabytes) of hard drive storage and ran at a central processor unit (CPU) clock speed of 4.077 megahertz (0.004077 gigahertz). By way of a modern comparison, the laptop used for writing this document operates at a CPU speed of 2.66 gigahertz, a 652.4 fold clock speed increase relative to the IBM XT. This medium-price range laptop also has an addressable memory space of 4 gigabytes, over 250,000 times larger than the 1981 IBM XT, and hard disk storage of 150 gigabytes, which is 15,000 times more disk storage space.

Not surprisingly-- the birth of heuristic optimization algorithms is inextricably tied to the rise of the microcomputer. Most certainly, the spread of microcomputers and their computational capability provided the essential tools for heuristic algorithm development. Conceptual approaches which had here-to-for been theoretical constructions, could be coded and tested. And they were. Not surprisingly, the description of many heuristic optimization algorithms dates back to this time. Examples include the development of genetic algorithms (1977), the description of particle swarm optimization (1995), simulated annealing (1983) and differential evolution (1995).

The cost of computer hardware, computer software and computer time no longer place an upper limit on the scale or scope of research agendas. The relaxing of these constraints has unleashed many different threads of research on heuristic optimization algorithms. Attitudes about computer resources used in research have also changed. Computational cost is now primarily a question of researcher patience. It is of little consequence to many researchers if their personal computer runs ten seconds, ten minutes or ten hours to achieve a solution. Improvements in the available computational tools, their low cost and near-universal availability have given rise to golden age of heuristic optimization research!

EAs in the Wild

Evolutionary algorithms (EAs) belong to a larger class of algorithms best described as being inspired by natural phenomenon, particularly the behavior of different organisms. These are often called nature based, nature inspired, or in some cases, biological algorithms. The universe of nature inspired algorithms is large and creative. Nature inspired algorithms span the realm from bacteria (Kim, Abraham and Cho 2007), to fireflies (Yang 2009), raindrops (Shah-Hosseini 2009), ants (Dorigo and Stutzle 2004) and beyond. Newly described algorithms appear in the literature on a regular basis. A selection of the more common and better documented nature inspired algorithms is shown in Table 2.

Table 2. Selected Nature-Inspired Optimization Algorithms

Algorithm	References
Ant colony optimization (ACO)	Dorigo and Stutzle (2004)
Artificial immune system optimization	Cutello and Nicosia (2002)
Bacterial foraging optimization	Kim, Abraham and Cho (2007)
Bee optimization	Karaboga and Bosturk (2007); Pham et al (2006)
Cuckoo algorithm	Yang and Deb (2009, 2010)
Differential evolution (DE)	Storn and Price (1995, 1997)
Firefly optimization	Yang (2010)
Fish optimization	Huang and Zhou (2008)
Genetic algorithms (GA)	Haupt and Haupt (2004)
Particle swarm optimization (PSO)	Eberhart and Kennedy (1995); Kennedy and Eberhart (2001)
Raindrop optimization	Shah-Hosseini (2009)
Simulated annealing	Kirkpatrick, Gelatt, and Vecchi (1983)

The evolutionary algorithms, including genetic algorithms, particle swarm optimization, and differential evolution are a sub-category of the nature inspired optimization algorithms. Evolutionary algorithms and their characteristics are the focus of this research and are discussed in greater detail in subsequent sections of this document.

Research on nature inspired algorithms is ongoing and active. There have been several evaluations and performance comparisons of nature inspired algorithms. These have typically focused on the less-esoteric members of this algorithm class. The most expansive of these evaluations is found in the book by Wahde (2008). Readily obtainable studies by Potter et al (2009) and Mezura-Montes and Lopez-Ramirez (2007) are also very useful contributions to this line of research.

Related Algorithms

Hybrid Algorithms

Hybrid evolutionary algorithms are frequently and routinely encountered in the applied literature. As distinct from memetic algorithms, which combine evolutionary algorithms and traditional (calculus based) algorithms, hybrid algorithms are constructed from two or more evolutionary algorithms. The resultant hybrid algorithm is often described as being (potentially) superior to either parent, especially in certain specific problem domains.

Hybrid combinations of nearly every evolutionary algorithm have been described. There are a plethora of hybrid combinations for PSO, DE, ACO and GA's and there are also hybrid combinations of other less well-known algorithms such as bee algorithms. Engelbrecht (2005) reviews several hybrid PSO algorithms including GA based PSO, DE based PSO (also see Liu, Cai and Wang (2008) and ACO based PSO. Banks, Vincent and Anyakoha (2008) review about 25 different hybrids and Neri and Tirronen (2010) cite about 30 more. A quick electronic perusal of the recent literature reveals a remarkable number of hybrid combinations and variants thereof.

At least some part of this activity may be driven by the need for researchers to differentiate their publication products. Even so, based on the existing number of different hybrids, this appears to be an incredibly fertile topical area for future research.

Memetic Algorithms

Memetic algorithms harness the global search characteristics of evolutionary algorithms with the fast local search properties of traditional (calculus based) optimization methods. Evolutionary algorithms such as PSO, DE and RCGA are able to rapidly and efficiently locate the neighborhood of the global optima, or a set of candidate optima. Typically however, their local convergence properties are rather slow. Evolutionary algorithms spend a disproportionate amount of time achieving convergence, after the neighborhood of the optima has been identified. Memetic algorithms utilize evolutionary algorithms to identify the neighborhood of an optimal point and then pass control of the optimization process to a traditional algorithm.

Engelbrecht (2005) reviews several PSO based memetic algorithms including hill-climbing PSO, stochastic local search PSO and gradient based PSO approaches. Additionally, there are a number of relatively comprehensive studies of memetic approaches. Particularly revealing are studies of GA based memetic algorithms (Li, Ong, Le and Gob 2008), DE based memetic algorithms (Neri and Tirronen 2010) and a comparison of different evolutionary based approaches (Nguyen, Ong

and Krasnogor 2007). Based on the available evidence, this two-step approach is quite efficient for continuous, differentiable functions and has the potential to stimulate many related threads of research.

EA Selection Process

Algorithm Selection

The selection of specific algorithms for this research was informed by the existing professional published and grey literature, applications to similar problems, performance reports and several practical considerations. As described in Table 2, the universe of evolutionary algorithms described in the literature is diverse and growing at a very rapid pace. Potentially, a number of different evolutionary algorithms could be applied to hydropower dynamic economic dispatch and unit dispatch problems. As with any research effort, this one was constrained by the resources available; primarily funding and researcher time. These and other practical constraints dictated, to some extent, the range and number of algorithms which could be explored.

Candidate Selection

An initial preliminary literature review was undertaken to identify candidate evolutionary algorithms for use in this research. The initial literature exploration was followed by a relatively extensive review of the power engineering literature with a focus on identifying intersections between the candidate algorithms and previous applications to electric power system problems. Subsequently, a more intensive review of the recent literature pertinent to specific candidate algorithms was conducted.

The literature review process resulted in identification of five candidate algorithms. These algorithms were; particle swarm optimization (PSO), genetic algorithms (GA), differential evolution (DE), ant colony optimization (ACO) and the Bees algorithm (BA).

Selection Criteria

Selection of evolutionary algorithms for this research project required some artistry and judgment. One factor which weighed heavily in the selection process was the depth and breadth of previous applications. The widespread use of a particular algorithm and the number of examples where it has been applied to a particular class of optimization problem provides some evidence of the algorithm's efficacy and potential for application in other arenas. For example,

both GA and PSO have been very extensively applied to an amazing variety of problem types. In contrast, the literature on the dragon-fly algorithm consists of a small handful of specific applications, the bulk of which are by the same author. With limited investigational resources to devote, the decision to eliminate the latter from consideration was rather straightforward

The hydropower dynamic economic dispatch problem and the unit dispatch problem are both examples of constrained optimization problems. For this reason, a substantial part of the decision process was focused on evolutionary algorithms which had been applied to the general class of constrained optimization problems. Although many of the evolutionary algorithms listed in Table 2 could potentially be modified, in some way, to accommodate constraints, it seemed prudent to limit the search to algorithms for which published examples existed. This eliminated some relatively promising algorithms from the subset of algorithms retained for further investigation. The Bees algorithm, for example, is a new and seemingly quite efficient evolutionary algorithm. Although they may exist, or be in process, no previous applications of BA to constrained optimization problems were uncovered during the literature search. As a result, this algorithm was not considered for further investigation.

Finally, the choice of evolutionary algorithms was further limited to those algorithms designed for the continuous real number domain. Although many applied problems can be specified in discrete forms (in fact, all continuous problems can be re-specified as discrete approximations), the most natural and appealing choice for solving a continuous real-valued problem is to use an algorithm which operates in the continuous real-valued domain. Ant colony optimization (ACO) is certainly a promising candidate algorithm, but is primarily useful in the discrete domain. For this reason, ACO was eliminated from further consideration.

Algorithms Selected

Based on the multiphase literature review, previous application to constrained optimization problems and limiting the choices to continuous real-valued algorithms resulted in a relatively small subset of evolutionary algorithms which were retained for detailed investigation. This subset includes; RCGA, DE and PSO. The lambda search (LS) algorithm, a traditional calculus based approach, was also chosen for use as a point of comparison. A short description of each of these algorithms follows while the details of these four algorithms are described more fully in Appendices 7, 8, 9 and 10.

Real Coded Genetic Algorithm (RCGA)

Genetic algorithms were the first of the evolutionary algorithms to be described in the literature. They use techniques inspired by evolutionary biology including inheritance, mutation, selection, and crossover. This research focuses on the less-

studied real coded genetic algorithm which is faster and more naturally applied to the dynamic economic dispatch problem, than the binary variant.

Genetic algorithms are based on virtual populations which are termed individuals (or phenotypes). For each generation or iteration, the fitness of every individual in the population is evaluated and the most fit individuals are selected and modified (recombined and possibly randomly mutated) to form a new population. The new population survives into the next generation or iteration of the algorithm. The algorithm terminates when a satisfactory fitness level has been achieved or the maximum number of iterations has occurred. Appendix 8 contains a complete description of RCGA.

Differential Evolution (DE)

Differential evolution (DE) was jointly developed by Storn and Price (1995, 1997) and is one of the more recently described global heuristic optimization methods. In many respects, it resembles a simplified form of genetic algorithm, albeit with several distinct and highly desirable performance characteristics.

The DE approach is based on a virtual population of np-independent individuals. During each generation, these individuals reproduce and undergo selection. Only the fittest individuals in the population survive to reproduce in the next generation. Over successive generations, the population becomes increasingly fit —thereby identifying the optimum (minimum or maximum) of a function. DE is described in considerably more detail in Appendix 9.

Particle Swarm Optimization (PSO)

PSO is a global heuristic optimization method. It was invented by Kennedy and Eberhart (1995) who developed the concept by observing the behavior of flocking birds. PSO is classified as a stochastic, population-based evolutionary computer algorithm for problem solving.

PSO utilizes np-independent virtual particles, which "fly" through the search domain, have a memory and are able to communicate with other members of their "swarm." Each particle has only one purpose—to identify the optimum (minimum or maximum) of a function within the feasible search space. PSO is described in more detail in Appendix 10.

Lambda Search (LS)

The lambda search (LS) algorithm is a traditional, calculus based approach, and its application to dispatch problems is rather well established and is described in Wood and Wollenberg (1996). The LS algorithm is arguably the fastest of the traditional calculus based methodologies which can be applied to this particular problem. It cleverly exploits the structure of this class of constrained optimization problem to reduce the number of decision variables to one (1). The LS algorithm is thus a univariate optimization approach and, as such, needs to identify the value of only a single unknown variable, rather than d-unknown

variables. Consequently, when it can be applied, it is very efficient. The LS algorithm was employed in this research effort primarily as a basis for comparing the selected EA algorithms. The LS algorithm and its application to the dynamic economic dispatch problem are described in Appendix 7.

Initialization

The first step in all of the heuristic optimization algorithms is to identify the starting positions of a specified number (np) of particles or individuals in the d-dimensional search space. This process is termed, "initialization."

Purpose and Implications

The choice of initialization strategy and its properties can profoundly influence the outcome of a heuristic optimizer. The successful identification of the global optima is dependent on the proximity of the initial points. To the extent an initialization approach does not adequately cover a particular region in the search space, and this region contains the global optima, the algorithm may fail to identify the global optima. Or, if the chosen initialization method places a number of particles in the region of the search space hosting a local optima, the algorithm may become trapped and converge on the local, rather than the global optima. Second, the number of iterations, the computational effort required and the convergence time required are related to the proximity of the initialized points to the optima. Finally, to the extent the initialization process is stochastic, the point of algorithm convergence, local versus global extrema and the precision of convergence will also vary.

In cases where the location of the solution is *a priori* unknown (which encompasses the majority of applied cases), it is desirable to distribute the np particles "equally" and "uniformly" within the search space. When the points are strategically distributed in this fashion, the probability that at least one point is close to the global maxima or minima is increased. On that concept, most researchers would agree. However, the mechanics of positioning a finite number of points in d-dimensional space such that they are approximately equally distributed is a nontrivial problem.

A 2-dimensional illustration conveys a considerable amount of information about this problem. Figure 4 illustrates the equal and uniform distribution of 49 points in 2-dimensional space. Some of the points are blue colored and some are open points colored yellow. In this figure, all of the points are positioned at each vertex of a lattice overlaid on the contour plot of a function whose maxima is identified with a green star. Most observers would agree these 49 points are equally and uniformly distributed in the bounded space and indeed, this can be

proven mathematically for this contrived case. The nature of the initialization problem becomes more evident if, for example, we limit the number of points to only the open yellow colored points, np=12. Devising a mechanism for allocating the 12 yellow colored points equally and uniformly in the search space is much more problematic. The complexity of this task grows immensely as the number of dimensions increases beyond the 2-dimensional example shown here.

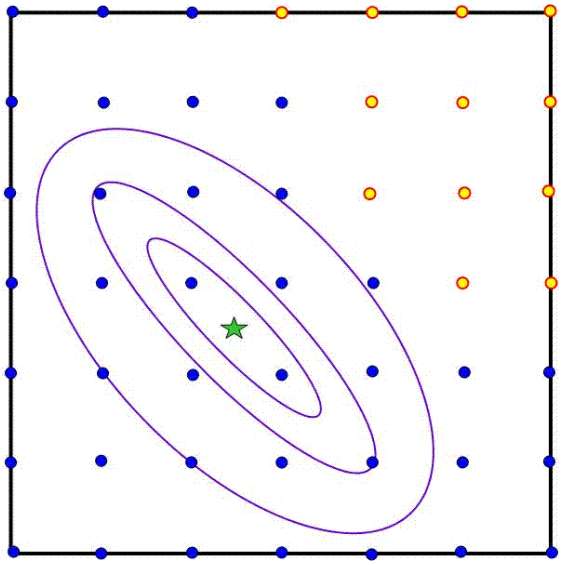


Figure 4. Lattice points in a search space (np=49).

Random

The vast majority of applied work employs the uniform random distribution to initially locate points in the search space. Figure 5 illustrates a random initialization of np=50 points in 2-dimensional bounded [1, 1] space.

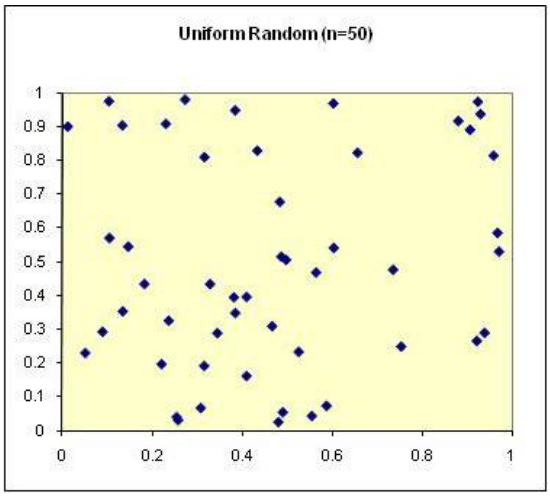


Figure 5. Uniform random initialization (np=50).

Visual comparison of Figure 4 (lattice points) with Figure 5 (random initialization) reveals some stark differences. In the random initialization

example (Figure 5), the points are considerably more numerous in some regions of the search area than in other regions. This is typical of random initialization methods, which often result in a non-systematic location of the initialized points within the bounded area. Clerc (2008) concisely describes this phenomenon in the first section title of a widely cited paper as, "Uniform Random Distributions: Easy but Bad."

One frequent contributor to poor random initialization performance is failure to employ a high quality random number generator (RNG). Random sequences produced by an RNG are an important component of this research. As described in Appendix 12, considerable effort was devoted to identifying and implementing an appropriate RNG for use in this research project. Interested readers are referred to Appendix 12 for additional technical information on this important topic.

Use of Sequences

In the last few years, some researchers have proposed the use of low discrepancy sequences for initialization purposes. Low discrepancy sequences are also called quasi-random or sub-random sequences. The points in these sequences are said to be more systematically located in the search space, with fewer gaps and more equal spacing between points. Low discrepancy sequences in the EA literature include the Sobol (Pant, Thangaraj, Singh and Abraham 2008), Van der Corput (Pant, Thangaraj and Abraham 2009) and Halton (Uy, Hoai, McKay and Tuan 2007) sequences, to name but a few.

Figure 6 displays a Weyl low discrepancy sequence initialization in the [1,1] space. It graphically illustrates the potential advantages of employing low discrepancy methods for initialization. Relative to the random initiation approach, low discrepancy sequence initialization can produce more uniform and systematic locations of points in the search space, with fewer gaps and more equal spacing between points. Low discrepancy sequences figured rather prominently in this research effort and are described more fully in Appendix 13. Appendix 13 also contains a useful comparison of random initialization with several low discrepancy sequences.

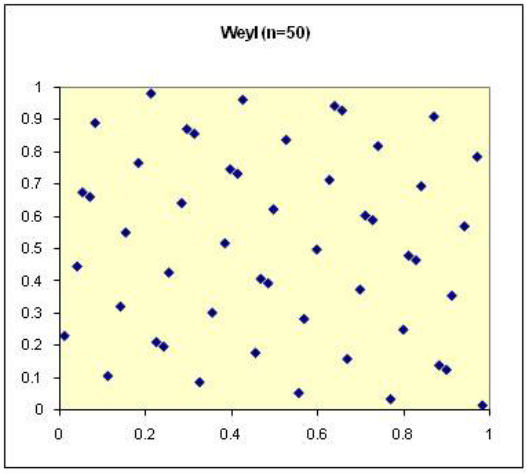


Figure 6. Weyl initialization (np=50).

Other

Several recent efforts have utilized other logical and mathematical approaches for initialization. A selection of these approaches include the simplex method (Parsopoulos and Vrahatis 2002), quadratic interpolation (Pant, Singh and Abraham 2009), tessellations (Richards and Ventura 2004) and the opposition method (Rahnamayan, Tizhoosh and Salama 2008, Omran 2009).

The simplex method is a rudimentary optimization technique developed by Nelder and Mead (1965). As described in Press et al (1989), it is slow, relatively robust and readily coded. As applied in this context, np particles are first randomly initialized in the search space. Each of these individuals is then used as a starting point for a user specified number of simplex iterations. The simplex algorithm moves the point towards a local or global extrema. As might be anticipated, this procedure improves the fitness of each particle *vis a vis* their randomly initialized positions. Collectively, the fitness of the initial points is improved. These improved starting positions improve the performance of the heuristic optimizer and accelerate its convergence. Like all of the approaches described here, the simplex method entails some additional implementation complexity and imposes some computational overhead.

The opposition method was originally described by Rahnamayan, Tizhoosh and Salama (2008) and is employed for initialization purposes by Omran (2009), who uses the term opposition based learning (OBL). The OBL procedure improves the starting fitness of the initialized points using an ingenious approach. Implementation of this approach is conceptually straightforward. First, np individuals are randomly initialized in the search space bounded by [a, b]. An additional np "opposite" points are calculated using the opposition equation shown in equation 11.

(11) $$p_o = a + b - p_i$$

This procedure results in a total population of 2*np individuals, or particles. The fitness of these 2*np particles are assessed and the particles are sorted by their fitness. The np most fit particles are retained and their positions are used to initialize or start the heuristic optimizer. Like the other approaches described here, the OBL method entails some additional implementation complexity and computational burden.

Previous research has focused on the application of different initialization methods to a suite of test problems. Their potential efficacy when applied to the solution of the hydropower problems examined here is unknown. Subsequent sections of this document will report the results of experiments which explore the use of several of these initialization techniques.

Constraints and Constraint Handling

Types of Constraints

Constraints separate the solution space into feasible and infeasible spaces. The resulting feasible solution space is generally limited and can be discontinuous, even when the optimization problem is not. Constraint equations can be linear or nonlinear in nature. In general, there are three classes of constraint equations. These broad classes are; boundary constraints, equality constraints and inequality constraints.

Boundary constraints serve to define the borders of the solution space. Boundary constraints commonly take the form of simple upper or lower bounds on the independent variables. These are called "box" constraints. However, simple bounds are not the only types of boundary constraints. For example, the circumference of a hypersphere (a sphere in multi-dimensional space) is also a boundary constraint. An example of a simple lower bound constraint is shown in equation (12).

(12) $$x_i > c$$

Equality constraints specify that a function of the independent variables is equal to a scalar constant. For example the amount of electricity generated (supplied) at a given instant must be equal to the amount of electricity demanded at that time. An example of an equality constraint is shown in equation (13).

(13) $$x_1 + x_2 + x_3 = k$$

Inequality constraints specify that a function of the independent variables must be greater than or equal to, or less than or equal to, a given scalar constant. For example, the contents of a reservoir must be less than or equal to the storage capacity of the reservoir. A simple example of an inequality constraint is shown in equation (14).

(14) $$v_t \leq f(x)$$

Constraint Handling Methods

Research on the incorporation of constraints in evolutionary programming methods and the solution of constrained optimization problems with evolutionary methods is rather voluminous. Carlos Coello Coello published a widely cited synopsis of this work (Coello Coello 2002) and maintains an online annotated bibliography summarizing this ever-expanding body of research (http://www.cs.cinvestav.mx/~constraint/index.html). In September 2010, this bibliography exceeded 89 pages in length.

Generalizing rather broadly from the literature, constraint handling techniques can be classified into six categories.

Problem Reformulation

An approach borrowed from traditional calculus based optimization methods is to reformulate or convert a constrained optimization problem into an unconstrained problem. For example, many constrained optimization problems encountered by engineers and economists can be solved by employing the method of Lagrange (see Appendix 5 for an example of this approach). The method of Lagrange is based on cleverly introduced artificial variables (such as λ) and slack variables. Using this approach some constrained optimization problems can be converted to unconstrained problems and readily solved. Other and more complex mathematical reformulation approaches can also be employed. For instance, Monson and Seppi (2005) describe a mathematical approach for projecting equality constraints onto a reduced (homomorphous) solution space. The reformulated unconstrained problem is solved and the values of the variables in the original problem can then be calculated.

Rejection of Infeasible Solutions

Unlike traditional calculus based optimization approaches, evolutionary algorithms carry multiple solutions through each generation and iteration. Perhaps the most direct method of ensuring that infeasible solutions are not used in the formulation of the solution in the next generation is to preclude them. Using this approach, potential solutions are first tested for feasibility. If a candidate solution is feasible, it can be stored as a solution and used as a basis to search for solutions in the next generation. However, if a potential solution is tested and found to be infeasible, it is not admitted as a solution and not used as a search basis in future iterations.

Penalty Approaches

A commonly employed constraint handling method is to mathematically disadvantage or penalize solutions which are infeasible. This approach is widely used both in traditional calculus based applications as well as in the application of evolutionary algorithms. In the evolutionary algorithm case, applied to a maximization problem, fitness (F) is often defined as the sum of the objective function value (f(x) *minus* the infeasibility penalties (P), if any as shown in (15).

(15) $$F=f(x) - P.$$

If we assume that a variable, say (x), is subject to a simple upper bound constraint, a penalty function may be defined as shown in equation (16).

(16) $$p = \begin{Bmatrix} if\ x >= k, & c + u(x-k)^2 \\ if\ x < k, & 0 \end{Bmatrix}$$

In this case, if x>=k, then the penalty is calculated as some constant (c) plus a scalar (u) times the square of the amount that x exceeds the upper bound (k). This

construction ensures that, (a) there is a penalty if x exactly equals the bound, k, and, (b) the magnitude of penalty increases rapidly as x exceeds the upper bound.

The penalty function approach is quite effective in evolutionary algorithm setting since it serves to disadvantage the set of solutions which are infeasible, relative to those which are feasible. In this context, the individuals with the greatest fitness form the basis for potential solutions in succeeding generations. As a result, the population will select for or move towards the feasible solution space with each new iteration.

The weakness of the penalty function approach is that specification of the penalty function (in the example above, the penalty function is specified as quadratic) and the magnitudes of the penalty function parameters (c and u in the example above) must be determined using judgment and experimentation. In cases where there are a large number of constraints, this can be rather problematic. Farmani and Wright (2003) describe a self-adaptive formulation which can overcome this problem, albeit at the cost of some additional complexity.

Feasibility Preserving Methods

An alternative approach is to identify a set of feasible starting solutions and then ensure that each candidate solution is itself feasible. Examples of this approach are described in Engelbrecht (2005) and Paquet and Engelbrecht (2003, 2007). In these sources, the authors introduce the Converging Linear Particle Swarm Optimization (CLPSO) algorithm. This algorithm solves an optimization problem with an arbitrary number of linear constraints.

Repair Methods

In this constraint handling approach, an operator or rule is used to construct a feasible solution from a solution which is infeasible. Consider, for example, an inequality constraint on the variable (x) which restricts x to be less than or equal to a scalar constant (k). An operator which corrects for (or repairs) an infeasible value of this variable might be constructed as shown in equation (17).

$$f(r \mid x) = \begin{Bmatrix} if\ x > k, & x = k \\ if\ x <= k, & x = x \end{Bmatrix} \tag{17}$$

Repair methods are widely used in evolutionary algorithms since they are both readily implemented and effective.

Mixed Approaches

In practice, most applications of evolutionary algorithms use a combination of all of the constraint handling methods described thus far. It is not uncommon for a particular application to reformulate some part of the problem to an unconstrained representation and to also employ penalty functions, repair methods and reject infeasible solutions. As shown in the literature, the artistry is in the identification of the most effective approach or combination of approaches for the efficient solution of a particular class of problems. As might be anticipated, the

recommended approach varies with the type of evolutionary algorithm employed and is almost certainly problem specific.

Fitness Comparisons with Constraints

Pair wise comparison of two solutions to identify which is the most fit is relatively straightforward for unconstrained optimization problems. In the case of constrained optimization problems, such comparisons are made considerably more complex because of the potentially confounding influence of the penalty function. To illustrate this problem more fully, first recall that when applied to a maximization problem, fitness (F) is often defined as the sum of the objective function value (f(x) *minus* the infeasibility penalty (P), if any.

(18) $$F=f(x) - P.$$

It may also be useful for the discussion which follows to recall that for constrained maximization problems, an infeasible solution is typically one in which one or more variables exceed their upper bound restrictions. By definition, the objective function value in these cases is higher than it would be if the solution were feasible.

Typically, the analyst may spend considerable time understanding the nuances of their particular optimization problem and judiciously selecting the values of the penalty function parameters. This systematic approach will help the analyst to properly scale the penalty function value in relation to the objective function values. Even so, identification of the "most fit" solution in a pair wise comparison remains problematic. The operative question in such cases being--does the (negative) value of the penalty function outweigh the objective function value? What if the penalty is rather small compared to the value of the objective function? In recognition of this logical and mathematical dilemma, most applications of evolutionary algorithms utilize an oft-cited work on this subject by Deb (2000).

Following Deb (2000), for any pair wise comparison of solutions, there are three possible cases. These are; (1) both solutions are feasible (the penalty is zero), (2) one solution is feasible (the penalty is zero) and the other solution is not feasible (the penalty is nonzero), and, (3) both solutions are not feasible (the penalties are both nonzero). Deb (2000) devised a comparison scheme for selecting the most-fit solution under each of these cases. This scheme is described in the bullet list which follows.

- If both solutions are feasible, select the solution with the greatest fitness. Owing to the fact the penalty is zero for both solutions; this is equivalent to selecting the solution with the highest value of the objective function.
- If one solution is feasible and the other solution is not feasible, then select the feasible solution without regard to its fitness value. It is useful to note

that under this criteria, the value of the objective function is not a factor in the decision process.

- If both solutions are infeasible, select the solution which has the lowest value of the penalty function (the most feasible solution). Once again, the value of the objective function does not play a role in this decision.

The selection scheme devised by Deb is straightforward and readily implemented in code. It has found wide-spread application and acceptance in evolutionary algorithms. Although it is not entirely foolproof, it is often used and (very) often cited.

Performance Measures

Algorithm Performance Metrics

Ascertaining the success of an evolutionary algorithm in identifying the solution to an optimization problem can be a rather subjective undertaking. Furthermore, discerning real, rather than apparent, differences between two evolutionary algorithms can be especially problematic. These difficulties arise for two disparate reasons: (1) the characteristics of real-life engineering problems, and, (2) the nature of evolutionary algorithms. Some further explanation will help to put both of these subjects in perspective.

Real-life Engineering Problems

Many, if not all, readers of this document are familiar with solving textbook example optimization problems. The majority of these problems are convex, and each has a single known optimal solution. Generally, the objective is to solve these find the optima of such problems, typically with a traditional, calculus based approach. Identifying the minimum or maximum point is often undertaken analytically, for relatively simple textbook problems. More complex problems are attacked with a variety of numeric methods, such as the Newton Raphson method referred to previously and described in Appendix 6. When the latter approach is utilized, the focus is to efficiently and reliably identify the optimal point to within some acceptable level of numeric precision.

In contrast to textbook optimization problems, many real-life engineering optimization problems have unknown optimal points. [If their solutions were known, there would be no need for algorithms to solve them]. To state the obvious point, there is no way to know when the optima has been found. Complex, ill-behaved problems with multiple local optima are relatively common in applied efforts. Algorithms may converge on a particular local optima, or may

converge on a different local optima when started from varying initial positions. This gives rise to a further complexity—identifying which, if any, of the identified local optima is the maxima or minima of the function.

Nature of Evolutionary Algorithms

The inherent nature of evolutionary algorithms can obscure the attainment of the optima and certainly makes it much more difficult to discern between two competing candidate algorithms. First, unlike calculus based solution approaches, evolutionary algorithms carry multiple solutions throughout the iteration process. For example, EA's carry one solution with each member of the population. In the population, some of these solutions are inferior solutions and one of them is the "best" solution. Furthermore, these solutions vary with each application of the algorithm. For any given algorithm trial, the np solutions are randomly initialized (the most common approach, see Appendix 12) within the search space. By random chance, some of the initialized solutions may land in the feasible solution region, or perhaps not. The specific initialization process and the initialization itself give rise to varying degrees of progress towards a solution. Likewise the stochastic nature of the solution algorithm, as manifested at each generation or iteration, has an influence on the algorithms rate of progress towards identification of the optima. For one trial, a series of fortuitously generated random values may result in a rapid convergence on the optima. For a different trial, a series of unfortunately generated random values may result in a failure to converge, a premature convergence, spurious convergence or a lengthy convergence to the optima. Consequently, evolutionary algorithms may return different solutions for multiple independent trials, even when applied to the same problem. Clearly, the convergence behavior of an evolutionary algorithm will vary with each trial or experiment.

Multiple Trial Approach

Owing to the complexities of real-life engineering problems and the inherent characteristics of evolutionary algorithms, a multiple or replicated trial approach is typically employed to gauge their success and compare efficacy between two candidate algorithms. A trial is one independent application of the algorithm to a specific problem. Typically a pre-set number of trials, for example 50 replications, are carried out on the same problem and selected measures of success are extracted for each of the trials. At each trial, a new initialization of the population occurs and a new random sequence is generated. In aggregate, the resultant measures of success then serve as a more appropriate and informative gauge of algorithm success. Formal statistical analysis of replicated success measures, compared across candidate algorithms, allows for reasoned selection of more effective algorithms.

Common Measures of Performance

Although there are many possible performance metrics, four measures are most commonly encountered in the literature. These are accuracy, reliability, robustness and efficiency. Other metrics including diversity and coherence are discussed in texts but seldom encountered in the professional literature.

Accuracy

As might be expected, solution accuracy is of paramount importance in assessing algorithm performance. In the case of functions with a known global best solution, accuracy is assessed as the difference between the best solution achieved by the algorithm and the known global best solution, at a particular number of iterations. For functions with unknown global optima, accuracy is the fitness of the global best solution attained by the algorithm over a given number of iterations.

If the accuracy of two different evolutionary algorithms is being compared, the usual practice is to compare this metric for the same number of function evaluations (FE's) rather than iterations. This is said to provide a better basis for comparison since some algorithms may require more per-iteration function evaluations than others, thus being more computationally intensive and, in the process, obtaining more information about the search space.

When the derivative of the function can be computed, derivative information can be used to assess the quality of the solution achieved. If the derivative can be computed, it ought to be zero, or very near to zero, at the optimal point identified by the algorithm. Of course, the derivative will be zero at any stationary point, including both global and local optima. For this reason derivative information is not entirely informative.

Reliability

Algorithm reliability is of great importance both to researchers and practitioners. The greater the certainty that an algorithm will (a) converge, and, (b) converge on the global optima—the more the more useful the algorithm is. For evolutionary algorithms, reliability may be assessed by measuring the percent of solutions which fall within an acceptable tolerance of the known global optima, for a given number of iterations. Or, when prior knowledge of the function is unavailable, as the percent of solutions which converge to a specified tolerance for some specified number of iterations. This metric is especially applicable to highly complex functions for which convergence is less common and somewhat less informative for better behaved functions for which convergence is routine.

Robustness

Robustness is a term used to describe the variance around a particular performance criteria. The variance around a success metric is a measure of dispersion. The smaller the variance over some given number of iterations, the more robust or stable the algorithm is judged to be.

Efficiency

Efficiency is a measure of the resource cost or effort incurred to achieve a solution with a desired level of accuracy. Efficiency is typically measured in terms of the number of iterations (or generations) required by the algorithm, the central processor time (CPU) time required, or the number of function evaluations (FE's) required to achieve a solution.

In the context of evolutionary algorithms, efficiency is a particularly relevant performance metric and is especially telling relative to traditional optimization approaches. Evolutionary algorithms are known for being computationally intensive and requiring relatively long computational efforts to achieve solutions. For functions which can be solved with traditional calculus based approaches, efficiency comparisons between traditional solution approaches and evolutionary algorithms are often rather revealing.

Algorithm Stopping Criteria

Introduction

The preponderance of numerical optimization algorithms are based on some sort of iterative or repetitive procedure. An important aspect of these algorithms is the design of intelligent convergence or stopping rules. These rules detect when the routines have converged on a solution, and then halt the iterative process.

The Trade-Off

The design of stopping rules necessarily requires an explicit trade-off between computational cost (a function of the number of iterations and hence, time) and solution accuracy. At best, numerical optimization algorithms can provide an approximation of the true solution vector, not the exact solution. In general, the numerical accuracy of the solution vector is improved with each succeeding iteration. Theoretically, a numerical algorithm can identify the exact solution in an infinite number of iterations. In more technical terms, these algorithms can be shown to achieve the true solution only asymptotically. Luckily, most research requirements can be satisfied by an answer that is "close enough" to the true solution and is available in a finite timeframe. Two interlinked questions emerge. How close is "close enough," and, what is an acceptable computational cost?

In many optimization applications, the scale and nature of the problem will suggest an appropriate level of accuracy. In many financial applications, for example, an absolute accuracy of $1.00 (the nearest dollar) or $0.01 (the nearest cent) is more than sufficient. In other cases, accuracy requirements are less clear cut. Almost all numerical methods texts include a discussion of this subject.

Interestingly enough, the specifics of computer hardware and software design limit the number of significant digits of accuracy which can be achieved. This places an upper bound on the how close is "close enough" question. Press, et al (1999), Judge (1998), among other, have useful discussions of these limitations on numeric accuracy. Press, et al (1999) has an especially useful discussion of this topic and the roles that data type, word length, register size play. As a rule of thumb, both Press, et al (1999) and Judge (1998) admonish the researcher not to specify an accuracy level greater than the square root of the machine accuracy. For most computers with a 32-bit word length, machine accuracy is around 3×10^{-8}. This suggests that a tolerance level (δ, ε) of around 1.7320×10^{-4} is about the best that can reasonably be expected.

In the early days of computer assisted research, computational cost was a much more important consideration than it is today. At the dawn of the computer age, research teams were quite literally charged for each millisecond of computer time they used. Because computer hardware was both expensive and rare, researchers paid for, or were allocated, a computer budget. Other researchers depended on the same hardware and research researchers dared not exceed their computer budgets, or severe sanctions were levied.

In modern times, the widespread availability of microcomputers, their speed and their relatively low cost, combine to make computational cost a less-important consideration. Computational cost is now primarily a question of researcher patience, rather than a funding issue. To most researchers, it is unimportant if the computer runs ten seconds, ten minutes or ten hours to reach a solution (as long as it does so). If long run-times are anticipated, it may prove convenient to schedule an overnight computer run. Some routine mathematical simulations are expected to take several hours, to a day or more to complete. A decade ago, computational costs of this magnitude were an unimaginable research luxury!

Calculus Based Criteria

Typically, convergence criteria for calculus based optimization algorithms are based on the first order conditions for an extrema—which require the first derivative to be equal to zero. In the multivariate optimization context, the first order conditions require the gradient vector to equal zero. As a practical matter, the norm of the gradient vector is evaluated to detect when this has occurred.

Judd (1999, p. 104) provides a concise and straightforward explanation of a two-fold stopping criteria or rule. First, a test is applied to identify whether or not the solution vector is changing significantly between iterations. Second, a test is applied to identify whether or not the first order conditions are met. This combined approach is shown in equations (19) and (20).

(19) $$\left\| x_{n+1} - x_n \right\| < \varepsilon(1 + \left\| x_n \right\|)$$

Where: n = iteration number
x = solution vector
ε = convergence criteria
‖m‖ = norm of the vector m.

Equation (19) compares the norm of the difference between solution vectors at two different iterations with epsilon (ε) times one plus the norm of the solution vector from the last iteration. This part of the stopping rule identifies whether or not the solution vectors achieved at two different iterations are the same, or approximately so. The formulation guards against division by zero and allows for the researcher to set some desired level of ε for detecting when this has occurred.

If the first part of the convergence test is satisfied, the next step is to see if the solution vector at iteration (n) satisfies the first order conditions for an optimum. As might be expected, this test focuses on whether the gradient vector is zero, or approximately so. This part of the stopping rule is described by equation (20).

(20) $$\left\|\nabla f(x_n)\right\| \le \delta(1+\left|f(x_n)\right|)$$

Where: n = iteration number
x = solution vector
Δf(x) = gradient of the function.
δ = convergence criteria.
‖m‖ = norm of the vector m.
|f(x)| = absolute value of the solution.

Again, this well-devised formulation guards against the possibility f(x)≈0 and a possible division by zero.

If both parts of the converge test (equations 19 and 20) are satisfied, the solution vector has converged to an approximate optimal point. If the solution vector (equation 19) has converged, but the first order conditions are not met (equation 20), the solution has converged, but not near an optima.

Convergence tolerances, epsilon (ε) and gamma (δ) are used to test when this rule is satisfied. These tolerances are set by the analyst. Both of these control parameters are commonly encountered in optimization routines and, as discussed in Judd (1999), Press et al (1998) and elsewhere, are limited by the ability of the computer platform to characterize real numbers.

Criteria for Evolutionary Algorithms

The stopping criteria employed for traditional calculus based optimization procedures are not applicable to evolutionary algorithms. There are two reasons for this. First, evolutionary algorithms are multiple solution methods; they carry a number of solutions throughout the iterative computation process. In the case of particle swarm optimization (PSO), for example, each member of the swarm stores its (own) personal best solution. If the swarm size is n=40, forty solutions are maintained and iteratively improved during the lifespan of the swarm. In contrast, calculus based optimization algorithms carry only a single solution throughout the computation process.

Secondly, the stopping or convergence criteria for traditional calculus based optimization approaches, not surprisingly, rely on calculus concepts (e.g. derivatives, gradients, hessians, etc). Evolutionary algorithms require only primitive mathematical structures, do not need and generally eschew advanced mathematical constructs, such as derivatives. In fact, their derivative-free nature is often touted as one of the advantages of these algorithms. Furthermore, evolutionary algorithms are often applied in situations where the underlying functions are discontinuous and ill-behaved. In these cases derivatives for the underlying functions either cannot be analytically derived, or simply don't exist. This makes it impossible to apply the stopping rules used in traditional calculus based optimization approaches.

An ideal stopping rule for evolutionary algorithms represents an acceptable trade-off between computational efficiency and the probability of detecting convergence on the true optima. At the same time, such a rule should minimize the likelihood of prematurely halting the iterations before the true optimal point is identified.

The preponderance of published articles found in the evolutionary algorithm literature employ the maximum number of iterations as a stopping rule. Using this approach, the algorithm proceeds until a pre-set maximum number of iterations have been completed-- then it halts. The "best" solution from the population of solutions is identified and then reported. The primary advantage of this approach is it is simple to implement. This stopping rule is frequently used to compare the behavior of alternative parameter settings and algorithm variants. The disadvantage is profound—the preset maximum number of iterations may or may not correspond to the number of iterations required for algorithm convergence. For example, if the maximum number of iterations is set at 1000 and convergence is achieved at 10 iterations, there are 990 unnecessary iterations. Conversely, if convergence does not occur until 5,500 iterations, the results returned for 1000 iterations will not reflect the optimal solution to the problem. Since evolutionary algorithms are stochastic, their rates of convergence vary in a probabilistic manner. As applied to a given problem, one trial may converge in 50 iterations and another trial in 120 iterations. Without prior knowledge of the problem's convergence behavior, there is no known technique for effectively

setting the maximum number of iterations. All of these factors argue against the application of this stopping rule—except for comparative purposes.

Figure 7 illustrates the convergence behavior for the 24-hour dynamic economic dispatch problem solved using differential evolution (DE). This plot shows the mean fitness by iteration for 50 trials. This is a maximization problem and the fitness improves as the number of iterations increases. As shown, large improvements in fitness are made initially with the majority of the improvements occurring in the first 1,000 iterations. Fitness improvements thereafter are made only slowly and at extensive computational cost, relative to the accuracy obtained.

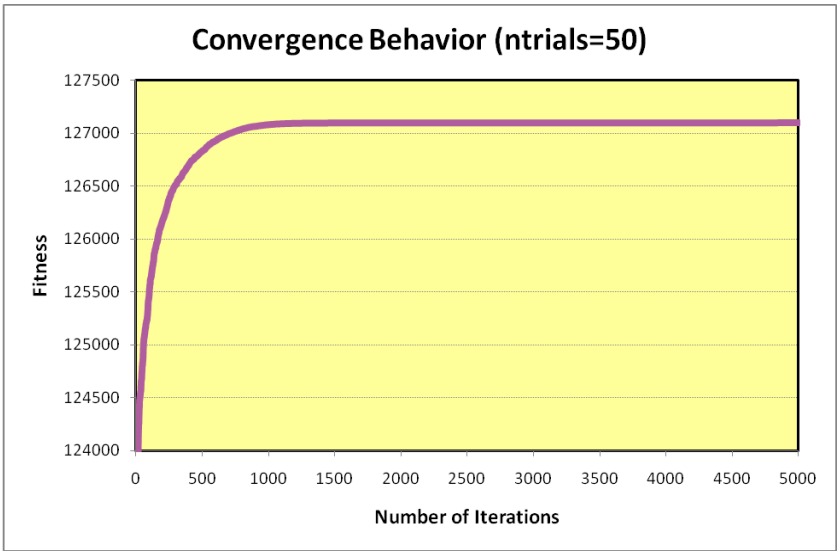


Figure 7. Convergence behavior with differential evolution.

As the number of iterations increases, the estimated solution asymptotically approaches the true solution. For this particular 50 trial experiment, the mean solution at 1,000 iterations is $127,079.25 and at 5,000 iterations, it is $127,097.14. This represents an improvement of about $17.89 (0.14%) at a cost of 4,000 additional iterations, which represents a 400% increase in computational cost.

The subject of stopping rules is not well addressed by the available texts such as Engelbrecht (2005), Kennedy and Eberhart (2001) on evolutionary programming. However some of the more recent research efforts have focused on this topic, for example Zielinski et al (2006), Zielinski and Laur (2007) and Zielinski and Laur (2008). The potential efficacy of the suggested approaches when applied to the solution of the hydropower problems examined here is unknown. Consequently, a non-trivial amount of effort was devoted to this subject as part of this research effort. Subsequent sections of this document will report the results of experiments which explore the use of several different stopping or convergence approaches.

Parameters, Tuning, and Variants

This section of the document describes the choice of parameter values and selection of the subset of algorithm variants examined in this research effort. The simple descriptor, “variants” is used to denote these algorithm variants throughout the remainder of this discussion. Considered in aggregate, these two subjects constitute a substantial portion of the literature devoted to evolutionary algorithms. As this is primarily an applied research effort, a less extensive and less systematic approach was employed.

Evolutionary algorithms have a relatively large number of parameters and approach variants. For example, the size of the population (np) is a user controllable parameter in all of the approaches examined here but each of these algorithms has additional parameters, some of which may interact with each other. Similarly, each of the evolutionary algorithms examined here includes user selectable algorithm variations, such as the neighborhood or global optimization strategies in PSO and the wide range of mutation strategies in DE. Selection of the appropriate value for these parameters and as well as choosing the particular logic, strategy and operational variants are specific to the logic of each evolutionary algorithm. Conclusions about the effects of parameter setting are mixed. Some researchers report effective applications of these algorithms are quite sensitive to parameter choice while others have suggested their efficacy is largely insensitive to the specific combination of parameter values chosen. Other researchers have reported that parameter choice is problem specific. On the topic of algorithm variants, the available evidence is also less than clear. The literature abounds with newly described variations for each of these evolutionary algorithms. Seemingly without exception, each of these variants is stated to dominate the other variants described in the previous literature.

Population Size

All evolutionary algorithms are multiple solution processes. The number of solutions, or population size, influences the performance of these algorithms and their successful application. There is an explicit tradeoff between the size of the population, the number of iterations required to achieve convergence and the computational effort. Many authors use the number of objective function evaluations (NFEs) as a measure of computational effort. For the PSO algorithm, for instance, the number of objective function evaluations required for each generation is given by the size of the population (np) times the number of iterations (iter) or, NFE = np*iter (disregarding initialization). For a problem of any given dimensionality (dim), the larger the population size, the more likely that one or more of the individuals in the population will be initialized to the vicinity of the global optima in the search space. All else being the same, a larger population might then be expected to use fewer iterations to converge more rapidly and converge on the global (rather than local) optima. The drawback to

large population sizes is that each member of the population must be evaluated at each generation. For complex objective function, with a larger number of dimensions, this can greatly increase the computational effort, requiring significantly longer solution times.

RCGA Parameters

The basic RCGA algorithm described in Appendix 8 has two parameters in addition to population size (np). These are reproduction probability parameter, also known as the crossover rate (χ) and the mutation rate parameter (μ).

As alluded to in Appendix 8, there are an amazing variety of reproductive variations in the realm of basic real coded genetic algorithm. The RCGA algorithm used here was restricted to the subset of possibilities wherein two parents produce either one or two offspring. Following implementation of one of the parent selection approaches, the reproduction probability parameter (χ) controls the likelihood the two selected parents will successfully reproduce. Typically, this parameter is chosen in the range of 0.50 to 1.00. A number of authors suggest setting this parameter from 0.90 to 0.95. Low values of the reproductive probability parameter or crossover rate (χ) effectively limit the genetic diversity in the population from one generation to the next. At the extreme, this can diminish the searching capabilities of population leading to a much more rapid and potentially premature convergence. The mutation rate probability parameter (μ) controls the rate of spontaneous genetic mutation in the offspring. Note that random mutations can be fitness enhancing or fitness degrading. This parameter controls the actions of any one of the various mutation schemes which may be employed in the RCGA. While the specifics of these mutation approaches differ in their details, high values of this parameter result in larger injections of genetic diversity in the population, increasing search behavior in the population. For complex or multimodal problems this can lead to a higher probability the global optima will be identified, naturally at the expense of convergence speed. For convex problems, this additional genetic diversity is primarily manifested as increased solution time and expense. A relatively lengthy review of studies on the effects of np, χ and μ on RCGA performance can be found in Haupt and Haupt (2004).

Table 3. RCGA Parameter Summary

Name	Abbreviation	Range	Setting Used
Population size	np	10 – 2*dim	40
Reproductive probability	χ	0.50 – 1.00	0.90
Mutation probability	μ	0.01 – 0.50	0.02

DE Parameters

The basic DE algorithm described in Appendix 9 has two parameters in addition to population size (np). These are the parent scale parameter (F) and the crossover (CR) parameter.

In the DE algorithm, the offspring or donor vector is constructed from three randomly chosen members of the population scaled by the parameter F. This parameter controls the importance of the parent traits, relative to those of the offspring. High values of F diminish the searching capabilities of population leading to a much more rapid convergence. This can lead to a higher probability of spurious convergence, or convergence at the location of a local optimum. Typically, this parameter is chosen in the range from 0.10 to 1.0. The crossover parameter (CR) controls the probability of crossover. If a randomly generated value exceeds the CR value, the parental trait is passed to the donor individual; otherwise the offspring trait is maintained. Greater values of the CR parameter have the effect of favoring offspring traits in the population, over succeeding generations. Relatively high CR values increase the range of search behavior in the population. For complex or multimodal problems this can lead to a higher probability the global optima will be identified, albeit at the expense of convergence speed. The CR parameter is typically chosen in the range from 0.10 to 1.0.

Table 4. DE Parameter Summary.

Name	Abbreviation	Range	Setting Used
Population size	np	dim – 3*dim	100
Population scale parameter	F	0.10 – 1.00	0.80
Crossover parameter	CR	0.10 – 1.00	0.30

PSO Parameters

The basic PSO algorithm described in Appendix 10 has two parameters in addition to population size (np). These are the cognitive weight parameter (c1) and the social weight (c2) parameter.

The cognitive weight (c1) parameter in the PSO algorithm controls the weight or importance of personal best information found by the individual particle itself, relative to the other members of the swarm. If the value of this parameter is relatively high, more weight or memory is accorded to locations in the search space that the individual particle has personally visited and less weight is given to information provided by the other members of the swarm. As a result, relatively high c1 values increase the searching behavior of the particle. For complex or multimodal problems this can lead to a higher probability that the global optima will be identified, albeit at the expense of convergence speed. The social weight

(c1) parameter in the PSO algorithm controls the weight or importance of social information found by the individual particle itself, relative to the other members of the swarm. The specifics depend on whether the neighborhood or global optimization strategy is employed. If the value of this parameter is relatively high, more weight or memory is accorded to locations in the search space which have been visited (collectively) by other members of the swarm and less weight is given to the particle's own personal best information. As a result, relatively high c2 values reduce the searching behavior of the particle, leading to a much more rapid convergence. For complex or multimodal problems this can lead to a higher probability of spurious convergence, or convergence at the location of a local optimum. For convex problems with only a single optima, rapid converge is a desirable characteristic.

Table 5. PSO Parameter Summary

Name	Abbreviation	Range	Setting Used
Population size	np	20 – 50, dim	20
Cognitive weight[1]	c1	1.0 – 4.0	2.80
Social weight[1]	c2	1.0 – 4.0	1.30

[1] The Clerc (2006) constriction factor, used in this effort, requires $c1 + c2 \geq 4.0$.

Variant Selection

Disregarding hybrid approaches (which are discussed elsewhere in this document) a wide range of variations on the basic evolutionary algorithms have been described, and are in use. To reiterate, the term "variants" is used in this document as a general descriptor for these. The number of variants seems to be proportional to the elapsed time since the algorithm was first described and seems limited only by aggregate researcher creativity and the need to differentiate research products for publication.

This effort benefited from a relatively extensive search of the pertinent literature completed previously (S&T Scoping Project ID Number 5992). This component of the study allowed for the admittedly subjective identification of the mainstream algorithm variants. One editorial aside-- several of these algorithm variants are considerably more complex than the underlying algorithms themselves. Some of the mainstream and potentially useful variants are described subsequently and were implemented for this research effort.

RCGA Variants

As a class, the RCGA and GA's exhibit the greatest range of variants on the basic algorithm. Disregarding the hybrid approaches (discussed elsewhere), there are an astonishing number of parent selection approaches, population survival

methods, mutation rules and crossover approaches, some of which are amazingly incredibly complex and computationally intensive. Haupt and Haupt (2004), Michelwicz (1996, 2010) and Peltokangas and Sorsa (2008) provide a relatively extensive sampling of these variants.

Parent selection in the RCGA is the method by which two parents are selected from the population for potential reproduction. Three parent selection methods were selected from the literature for use in this research effort. They are the random parent selection method (Random2), the four person tournament method (Tournament4) and the two person tournament approach (Tournament2). Under the Random2 method, one individual is selected systematically from the population as a whole and a second (different) individual is selected randomly from the population. The two selected individuals are then available for potential reproduction (crossover and mutation). The tournament4 approach selects four different individuals from the population as a whole. Two of these individuals then compete and the one with the greater fitness becomes a semifinalist. The two other individuals then compete and the one with the greater fitness enters the semifinals. Finally, the two semifinalists compete, and the one with the greater fitness wins and is available for potential reproduction. The tournement2 approach selects two different individuals from the population as a whole. These individuals then compete and the one with the greater fitness becomes a potential parent.

Crossover is the mechanism by which two potential parents exchange genetic material to create one or more offspring. Crossover in the RCGA context differs considerably from the binary GA case and numerous approaches have been developed to simulate this process. In the context of RCGA, the arithmetic crossover (Arithmetic) approach (Michalewicz 1996), the Laplace crossover (Laplace) approach (Deep and Thakur 2007), the linear crossover (Linear) approach (Wright 1991) and the heuristic crossover (heuristic) approach (Michalewicz 1996) were implemented for this research effort. The Laplace crossover approach is described in detail in Appendix 8. The linear (Linear) crossover approach produces three offspring using an extrapolation approach. An extrapolation weight, often 0.50, is employed. Any variable straying outside the feasible search domain is either censored or the solution is discarded-- the two offspring with the greatest fitness are retained. The uniform arithmetic (Arithmetic) crossover approach is often attributed to Michelwicz (1996). This approach uses a randomly generated value (0, 1) to form a set of weights (α, 1-α) which are then used to create a linear combination of the parent genes. A variant of this approach utilizes a different random value (and hence weight) for each choice variable represented in the parent genetic material. The heuristic crossover approach (Heuristic) was also developed by Michaelwicz (1996) primarily for use in constrained optimization problems. The heuristic approach generates a possible offspring from a randomly weighted differencing of the parent's genetic material, added to the superior parent's existing genetic material. If the offspring lies outside of the feasible domain, a new random weight is generated until a feasible solution is obtained.

Mutation helps to ensure genetic diversity is maintained in the population and some of the mutation variants described in the literature are quite ingenious. For purposes of this research effort three mutation approaches were selected and implemented. These approaches include the Gaussian mutation approach (Gaussian), the nonuniform mutation (Michalewicz 1996) approach (Nonuniform), and the uniform mutation (Uniform) approach (Michalewicz 1996). The nonuniform mutation approach is described in Appendix 8 and is not further described here. Under the Gaussian approach, a normally distributed random variable is added to the gene selected for mutation. This approach is relatively simple and effective in many applications although it does require the variance of the distribution to be specified, in some manner. Under the Uniform approach, genes have an equal probability of mutation. A gene selected for mutation is replaced with a uniform random value generated within the feasible search domain. This approach has two advantages. First, it is relatively easy to implement. Second, it executes rapidly.

Survival or recruitment, sometimes also known as replacement, is the process of determining which individuals from the offspring population and the parent population will survive into the next generation. There are a wide variety of recruitment approaches, which have evolved over time (see Reeves 2010 p. 71 for a summary). For purposes of this research effort, three survival approaches were selected from the literature and implemented in code. These are the traditional (Traditional) approach, the Elite_1 approach (Bucknall 2002) and a more general characterization of the elite approach, the elite np (Elite_NP) approach.

The traditional approach to recruitment is fairly straightforward—only the offspring survive into subsequent generations. While easily implemented in code, there is a distinctive logic flaw inherent with this approach. In the traditional approach there is a probability the individual with the highest fitness will be eliminated from the gene pool, slowing the evolutionary process and the search for an optima.

The Elite_1 approach preserves the genetic material from the fittest individual in the gene pool from one generation to the next. In the Elite_1 recruitment approach, the parents are ranked from highest fitness to lowest fitness and the offspring are ranked from highest fitness to lowest. The parent individual with the highest fitness (the Elite_1) replaces the lowest ranked offspring, provided it is of superior fitness. The remaining offspring and the Elite_1 individual, survive into the next generation.

There are many potential variations on the elitism approach. Conceptually, the retained elite fraction could vary all the way up to NP (here assuming a constant population size is maintained). For purposes of this project, the Elite_NP approach was employed. Under this approach all of the parent and offspring individuals are pooled and then sorted by fitness. The most-fit NP individuals from the pool are then retained and survive into the next generation, the less fit individuals are removed from the gene pool. This approach greatly increases convergence speed, often dramatically. Unfortunately, the genetic diversity of the

population diminishes as well and the likelihood of spurious convergence, or convergence at a local optimal point, increases.

DE Variants

Like the other evolutionary algorithms explored here, there are a number of variants on the basic DE algorithm. Disregarding the hybrid approaches (discussed elsewhere), there are a large number of mutation rules and crossover approaches, some of which are amazingly ingenious. Many of these build upon the seminal DE paper (Price and Storn 1995, 1997) which described 23 crossover and mutation combinations. Over time, a shorthand approach for describing and categorizing the more mundane of these variants has evolved. The notation DE/x/y/z is often used for this purpose. In this notation, x is used to specify the vector to be mutated which can be "Rand" (a randomly chosen member of the population) or "Best" (the member of the population with the current best fitness), y represents the number of difference vectors used, and, z denotes the type of crossover scheme employed. The most common crossover variant is the "Bin" or binary crossover approach.

For purposes of this research effort, six different crossover and mutation approaches were selected. These include the originally described DE/RAND/1/BIN and the DE/BEST/1/BIN approaches, but also include some of the more promising and exotic approaches such as the random scale factor (DERANDSF) approach, the trigonometric (TRIGON) approach, the time varying scale factor approach (DETVSF) and the self adaptive (SELFADAPT) approach (Brest et al 2006 version). These variants were sufficiently represented in the mainstream literature to warrant further investigation.

PSO Variants

Many examples of PSO variants can be found in the literature, the majority of which are reviewed in Valle et al (2008). Disregarding hybrid approaches (discussed elsewhere in this document) the two enduring variants appear to be the application of global or neighborhood optimization strategies. In the global optimization strategy, crossover is a linear combination of the best fitness value found by any of the members of the swarm (globally) and a particle's personal best fitness. This optimization approach results in faster convergence but also decreases searching behavior and increases the likelihood of spurious convergence or identification of a local, rather than global, optimal point. This optimization strategy should be distinguished from the neighborhood (or local) optimization strategy. In the neighborhood optimization strategy, crossover is a linear combination of the best fitness location identified by any of the members of a particle's neighborhood and the particle's own personal best fitness value. Figure 8 illustrates a globally connected swarm of np=8 (Panel A) and an np=8 swarm with a 3-member neighborhood structure or topology (Panel B).

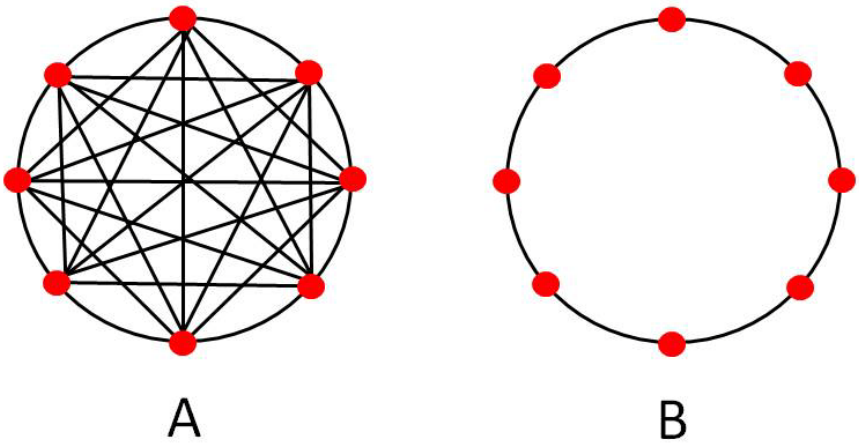


Figure 8. Globally connected (A) and 3-neighbor (B) swarms.

The use of a neighborhood structure serves to limit the information about the search space available to any single member of the swarm. Neighborhoods can, and are, constructed in a variety of shapes or topologies and follow an amazingly creative set of behavioral rules (see Kennedy and Mendes (2002) for some of the details). For purposes of this research, a star-type neighborhood topology, limited to five total members (including the particle itself) was employed. These 5-member neighborhoods limit the overlap or interconnection in the swarm. With each succeeding generation, information about the location of potential optima effuses from neighbor to neighbor, and from neighborhood to neighborhood within the swarm. This approach results in enhanced searching behavior, slower convergence times, and it reduces the probability of spurious convergence and convergence on local optima within the search space. The 5-member star-type neighborhood configuration proved to be relatively straightforward to implement in code and highly effective in application.

Following the literature review component of this research effort, Clerc's constriction coefficient (2006) was selected for use in the PSO algorithm. This PSO variant is described more completely in Appendix 10.

Development Process

This research project required an extensive behind the scenes software development effort. For the most part, the evolutionary algorithms examined in this research effort are rather new and certainly not commercially available. A relatively large-scale and time consuming development effort was required to make them operational.

Development Platform

Borland Developer Studio 2006 for Windows, an object oriented rapid application development (RAD) environment was employed for coding and construction of test platforms. This development environment is designed for use on Microsoft Windows 32 Kb operating systems, such as Windows XP. Although the development environment has web, web application, C, C++ and .NET capabilities, the Delphi language was used throughout this project. Delphi is an object oriented language which evolved from the Turbo PASCAL language, popular in the early 1980's.

The Borland Delphi 2006 compiler produces native Windows 32 Kb executable code. This environment eases the development of Windows based graphical user interfaces while allowing full code control. Of particular advantage for this project, the development environment includes visual component libraries (VCLs) for advanced graphics and database integration. These VCLs along with others for printer and device control, disk file operations and interfacing with the windows environment are implemented with "drag and drop" functionality in a fully visual integrated development environment. This greatly streamlines the development of Windows based applications allowing the researcher/developer to devote their resources to code development. The integrated graphics capabilities were a major consideration in the decision to deploy this platform.

Other development platforms were reviewed for possible use in this project. These platforms included MATLAB (www.mathworks.com), a commercially available package widely used in engineering applications, the open source European equivalent, SCILAB (www.scilab.org) and the open source general purpose statistical package, R (www.r-project.org). However, none of these platforms appeared to offer the ease of development, stability and integrated graphics capabilities required for this effort.

Three Phases of Development

Following the selection of candidate algorithms, a three phase development process was undertaken. First, the algorithm was coded and tested on three unconstrained test problems. Second, the algorithm was coded and tested on the dynamic economic dispatch problem. Third, a testing environment was developed for each algorithm.

Phase 1—Development with Test Problems.

Working from pseudo-code, flow charts, verbal descriptions in journal articles, code snippets and in some rare cases, translating from purportedly functional C source code, the selected algorithms were coded in Delphi, debugged and brought to a operating state. The three unconstrained three dimensional (3-D) test problems described in Appendix 14 were used during this phase to debug, and

more importantly, test the functioning and solution behavior of the coded algorithms.

A graphical user interface (GUI) was developed for each application. These GUI's (naturally) share a number of common features and functionality. Shared features include a tabbed page for selecting an initialization strategy, a tabbed page for selecting a convergence strategy and convergence tolerance and a tabbed page for controlling visualization.

Each GUI also has an Algorithm tabbed page which is customized for each algorithm. This customized tabbed page allows for easy user control of parameters specific to the algorithm and allows different variations of each algorithm to be selected. For example, the Algorithm tabbed page for the DE algorithm allows the user to select from a list of Mutation strategies, select the number of individuals in the population (np), set the value of the scale parameter (F) and select the value of the crossover (CR) parameter. In contrast, the RCGEN algorithm tabbed page allows the user to set the number of individuals in the population (np), select a parent selection strategy, select a crossover approach, select a mutation strategy and select a recruitment approach.

All of the applications share a common output GUI configuration, shown in Figure 9. Each application has a numerical output window and a graphical output window. The latter allows for real-time visualization of solution progress, a feature which has proven to be invaluable.

The behavior of the algorithms using different parameter settings and optional variants was observed both numerically and visually by judicious application of the integrated graphics capability. Figure 9 illustrates the graphical output screen of the RCGA program at iteration 6 during a solution of the Alpine function. This figure shows the plan view of this relatively complex function (see Appendix 14 for further details about this and other test functions). In the figure, the blue diamonds illustrate the (x,y) locations for each of the np=40 individuals in the population. The single red diamond located in the upper right-hand quadrant indicates the location of the optimal solution in the bounded search space.

The integrated graphics allows the researcher to observe the solution behavior in real-time while simultaneously monitoring the algorithm's numerical progress toward a solution. Progress metrics are written to the status bar at the bottom of graphics window. As reported in the status bar, shown in Figure 9, this plot is for the sixth iteration, the most fit individual in the population has a fitness (Fit) of 7.786E+000, the standard deviation (SD) of population fitness is 2.225E+000 and the visual delay (Del) is set to 100 milliseconds.

Implementation of these evolutionary algorithms involved overcoming a number of technical travails. This included selecting and developing a random (pseudorandom) generator, the use of low discrepancy sequences, the development of appropriate convergence or stopping criteria and the development and application of constraint and constraint handling methods.

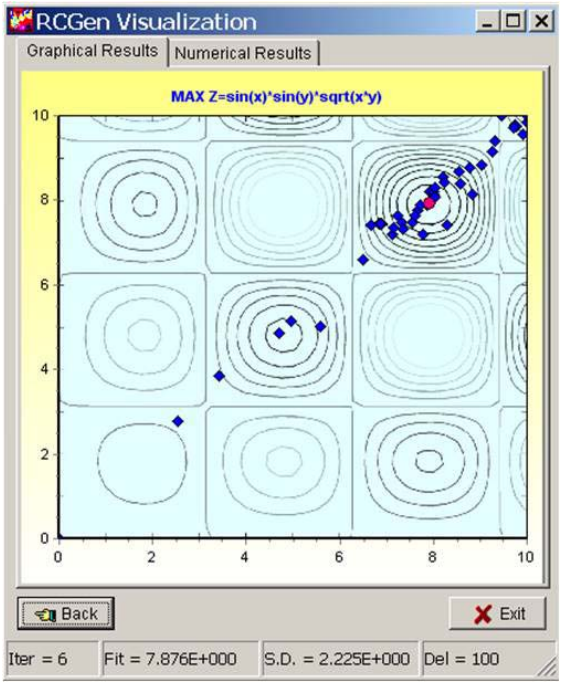


Figure 9. RCGEN program solving the alpine function.

Implementation of these evolutionary algorithms involved overcoming a number of technical travails. This included selecting and developing a random (pseudorandom) generator, the use of low discrepancy sequences, the development of appropriate convergence or stopping criteria and the development and application of constraint and constraint handling methods.

Phase 2—Economic Dispatch Problem

Working from the code base developed in Phase 1, the evolutionary algorithms were adapted for solution of the hydropower dynamic economic dispatch problem with constraints. The economic dispatch problem described in Appendices 3 through 5 was used during this phase to debug and complete initial tests on the coded algorithms. The economic dispatch problem is a constrained optimization problem and accommodating this class of problem required a further and rather extensive coding effort in its own right.

The graphical user interface (GUI) developed in Phase 1 was modified to accommodate the dynamic economic dispatch problem. Specifically, additional tabbed pages were added to the existing GUI's to allow for the added complexity of this problem class. Added GUI features included a tabbed page for selecting seasonal avoided cost (price) data (summer or winter) and for selecting either a 1-day (24 hours) or 1-week (168 hours) analysis period, a tabbed page for controlling the constraints on minimum and maximum release rates, ramp rates and the amount of water scheduled for release during the analysis period. Additionally, a tabbed page was added to allow for more detailed monitoring of the numeric progress towards a solution.

All of the common output GUI's were modified to better suit the dynamic economic dispatch problem. The graphical output for this application was modified, as shown in Figure 10, to illustrate the minimum and maximum constraints and display the optimal hourly pattern of generation and release. The numerical output window was revised to show the hourly details of the optimal solution for this problem. An additional numeric output window was added to record selected intermediate output metrics for each iteration (or generation) as the algorithm evolved towards a solution. This proved to be an invaluable debugging aid. Figure 10 illustrates the graphical output for a default solution of the dynamic economic dispatch problem solved for 1-week (168 hours) during the summer season.

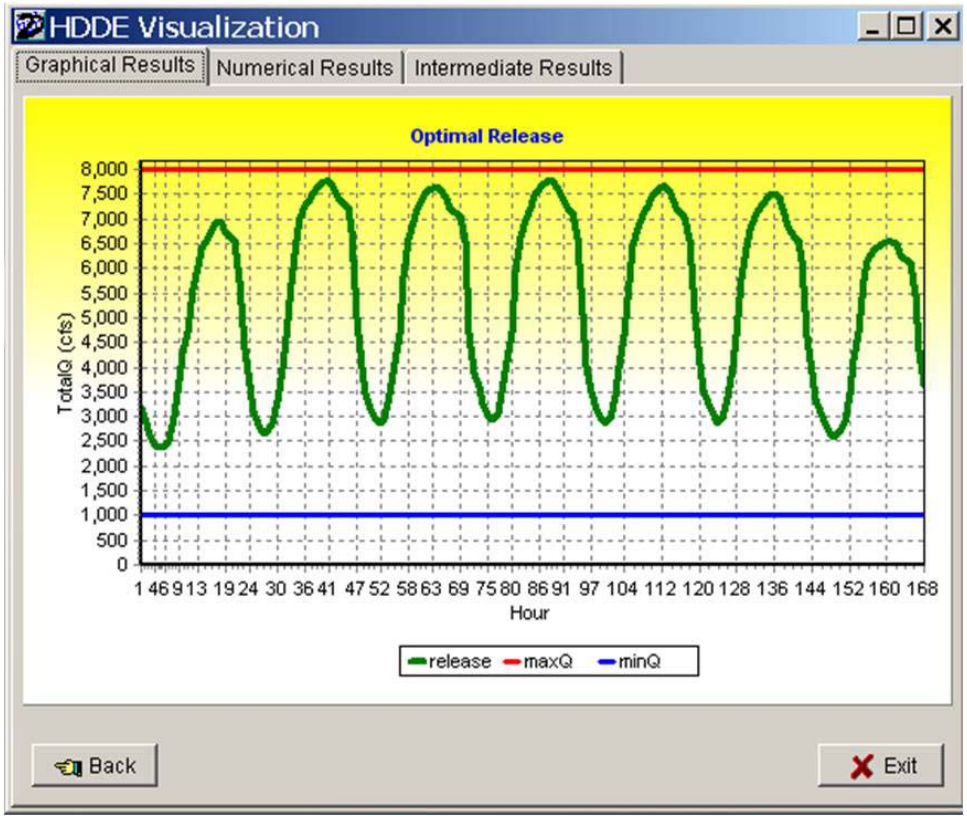


Figure 10. HDDE solution to 168-hour economic dispatch problem.

Phase 3—Testing Environment

The purpose of Phase 3—development of a testing environment, was to construct a framework for the unattended replication of experiments while saving success metrics, performance measures, numerical outcomes and other summary data for subsequent statistical analysis. As described previously in this document, evolutionary algorithms are stochastic in nature. For any given set of starting values, an algorithm may achieve a different, slightly different, or vastly different solution. Or, it may fail altogether. This range of potential outcomes arises

because of (a) the initialization approach employed, (b) the random underpinnings of their solution behaviors, and (c) the approach implemented to detect convergence on a solution. Consequently, a single successful solution, while indicative, is by no means conclusive evidence of the successful application of one of these algorithms. In the context of evolutionary algorithms, replicated trials followed by statistical analysis are required to support even minimal conclusions about their suitability for a specific class of problem.

Phase 3 required using the code base developed in Phase 2, and adding additional code to allow for repeatedly running the algorithm and recording salient success and performance measures for each run. Relative to the development effort expended in Phase 1 and Phase 2, this was readily accomplished and required only minor modifications to both the input and output GUI's. However it required only limited (additional) code development. The testing environment developed in this, the final Phase of the development effort, was utilized to produce the replicated experimental results described subsequently in this document.

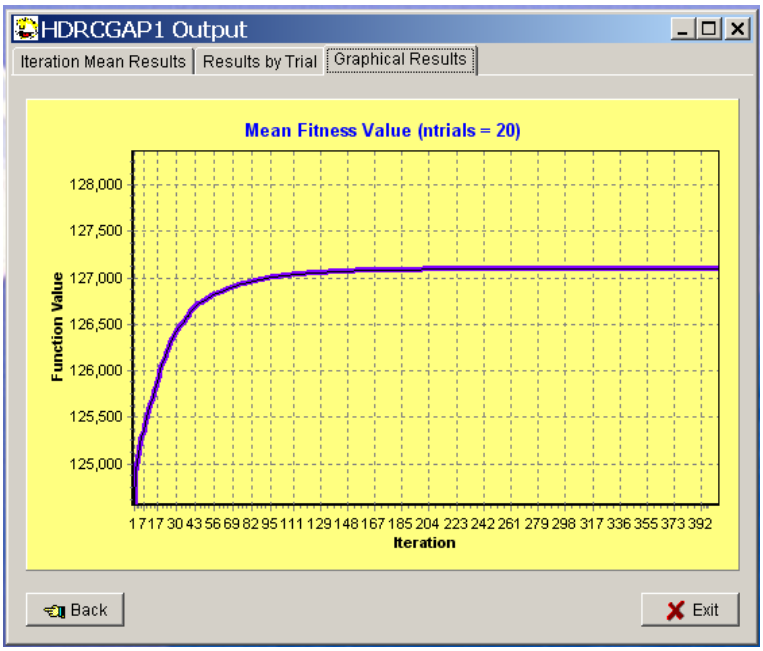


Figure 11. Test environment graphical output.

Experiments Undertaken

Given the number of parameters, options, algorithm variants, input vectors and problem features described, there are a very large number of experiments could be undertaken. While a comprehensive effort would surely be a valuable contribution to the state of knowledge, resource limitations dictated that only selected experiments be completed and reported in this document. The

experiments which are described here were selected primarily to provide insights about the applicability of EA's to the dynamic economic dispatch problem, their performance and the factors which might influence this decision

Initialization Approaches

Review of the pertinent literature revealed a number of efforts which reported systematic differences in EA performance resulting from initialization method. The literature on this topic was both extensive and conclusive—the use of the uniform random approach for initialization was judged to be inferior to other approaches. Based on this body of literature, considerable researcher effort was allocated to developing, testing and applying promising alternative initialization approaches to the hydropower problems examined in this research effort. This required the testing and development of four low discrepancy sequences including the Niederieter, the Habor, the Weyl/Torus and the Halton (see Appendix 13 for additional explanation) as well as the opposition based learning (OBL) method.

After coding and validating the functioning of these different initialization approaches, a systematic set of performance experiments was undertaken. For purposes of the replicated initialization experiments described here, 50 trials were undertaken on the 24-hour constrained dynamic economic dispatch problem (24-hour, problem dimensions = 24) using the summer price vector described in Appendix 15. To ensure a valid comparison across algorithms, the population (swarm) size was set at 50 individuals for all of the evolutionary algorithms (np=50). It should be noted the performance of some of the EA's, such as PSO and RCGA may be disadvantaged by setting the population size to this level for this comparatively small dimension problem. A common stopping rule, the Elite_SD rule (with tol=1.0e-04), was employed for all of the replicated experiments.

Table 6. Initialization Approaches — Experimental Results

Approach	RCGA		DE		PSO	
	Mean Iter	Mean CPU time (Sec)	Mean Iter	Mean CPU time (Sec)	Mean Iter	Mean CPU time (Sec)
Random	494	0.195	242	0.115	447	0.506
Neiderieter	506	0.202	245	0.116	452	0.517
Weyl/Torus	500	0.199	242	0.114	442	0.497
Habor	495	0.194	241	0.114	438	0.496
Halton	471	0.185	243	0.114	443	0.500
OBL	491	0.193	244	0.116	446	0.504

Table 6 summarizes the results of the replicated initialization experiments. The results shown in this table are both unremarkable and unexpected. For the constrained dynamic economic dispatch problem, there appears to be no discernable difference between the uniform random initialization approach and

any of the other approaches. While this result appears to contradict the results reported in many earlier studies, none of the preceding studies focused on this particular type of constrained optimization problem. Scrutiny of this table suggests the Halton Sequence may provide small performance gains for this problem. However no statistical analysis has yet been undertaken to confirm or refute this observation.

Stopping Rules

Consistent with the philosophy of evolutionary programs, a convergence or stopping rule should utilize the fitness information available at each iteration, be simple and effective. In keeping with this theme, one approach is to use the population mean and standard deviation for detecting convergence. Operationally, the mean and/or standard deviation of the solutions found by all of the individuals in the swarm or population are calculated for each iteration. When these metrics change by less than a pre-set tolerance, or fall within an acceptable tolerance, the algorithm has converged on a solution. The advantage of this method is that it is relatively easy to implement, is problem and scale invariant and is brutally effective. Arguably, this approach may be overly conservative and computationally inefficient often requiring an extensive number of iterations for all of the members of a swarm or population to converge on the optimal point.

Zielinski et al (2006), Zielinski and Laur (2007) and Zielinski and Laur (2008) explore the subject of convergence rules for particle swarm optimization (PSO) and differential evolution (DE). They examine single objective problems with different dimensions using varying population sizes. In aggregate, the authors systematically explored the performance of a suite of approaches, some of which were quite esoteric. They recommend two methods for use with PSO and two methods for use with DE. They suggest a variant of the standard deviation approach be examined more fully in future research efforts.

Motivated by the work of Zielinski and Laur (2008), two additional convergence criteria were developed and investigated in this research effort. These were the elite mean (Elite_Mean) and elite standard deviation (Elite_SD) approaches. These two approaches are based on the observation that one or more members of the swarm or population will identify the optimal point well before the other members of their cohort. We will call the portion of the population which converges rapidly, the “elites.” At each iteration, uninformed members of the cohort will continue to search in the solution space, sometimes far from the optimal point. It can, and often does, require many additional iterations for all of the members of a swarm or population to converge on the optimal point, previously discovered by a few individuals. Observations made in the early phases of this project suggest that a disproportionately large number of iterations are required to produce convergence in the last quartile of the swarm or population.

The elite mean and elite standard deviation stopping rules are based on the empirical observation that a subset of particles (the elites) converge very rapidly and a subset of the remaining individuals converge extremely slowly. To take advantage of this, calculation of the elite mean and elite standard deviation convergence metrics are based solely on the behavior of the elite or best performing particles. The behavior of the lower performing individuals, which potentially could take many more iterations to converge, is ignored.

Identification of the elite members of the swarm or population is, of course, somewhat problematic for the purposes described. Since we do not know *a priori* the optima for a given problem, we cannot know with certainty if say, the two most fit particles in iteration number 561 have converged on the solution, or not. If the elite proportion of the population were defined as the most-fit 5%, the potential for spurious convergence may be quite high. Alternatively, defining the elite proportion of the swarm as the most-fit 99% may result in significant and unwarranted computational cost. This choice represents a fundamental analysis trade-off which is to some extent arbitrary, but is surely problem dependent.

During this research some exploration of this trade-off was undertaken. This exploration could not be described as either comprehensive or conclusive. However, it was sufficiently extensive to make some inferences about the application of these stopping rules to the types of problems examined here. After some experimentation, the elite proportion of the population was defined as that 90% of the population or swarm which was most fit. By definition, the individuals classified as elites varied dynamically from one iteration to the next. Using this definition for the elites resulted in excellent computational performance with very little likelihood for spurious or premature convergence. These stopping rules are relatively easy to implement, do not add extensive computational overhead and proved to be very effective for the types of optimization problems we examined.

For purposes of the replicated stopping rule experiments, 50 trials were undertaken on the 24-hour constrained dynamic economic dispatch problem (24-hour, problem dimensions = 24) using the summer price vector described in Appendix 15. To ensure a valid comparison of the different stopping rules, the population (swarm) size was set at 50 individuals for all of the evolutionary algorithms (np=50). It should be noted that some of the EA's, such as PSO and RCGA may be disadvantaged by setting the population size to this level for this comparatively small problem. For applicable "elite" approaches, the elite fraction was set to 0.90. In all cases the convergence tolerance (ctol) was set at 1.0e-04.

Figure 12 shown below compares the performance of four different stopping rules on this same problem. It compares the mean central processing unit (CPU) time, measured in seconds, required for convergence over 50 trials between the maximum iteration approach (maximum iterations = 5000), the population standard deviation (Pop_SD) approach, the elite standard deviation (Elite_SD) approach and the elite mean (Elite_Mean) approach.

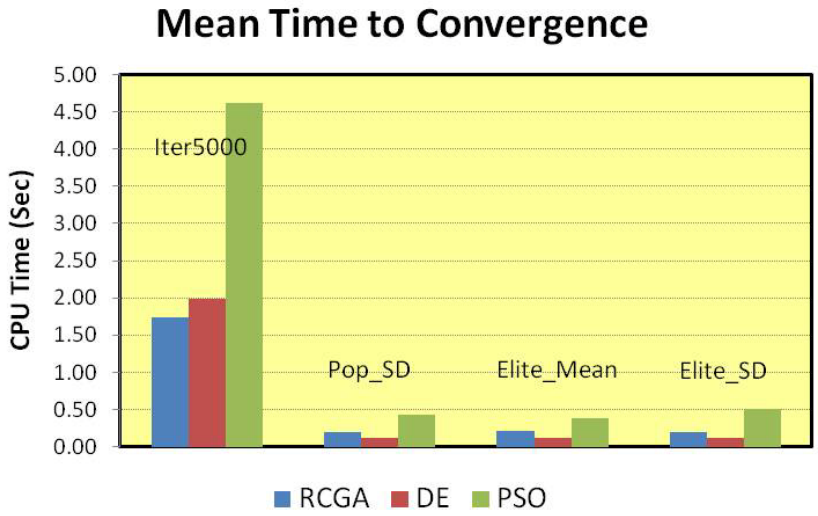


Figure 12. Results of Stopping Rule Experiments (dim=24).

As shown in this figure, there is a large difference between the mean computational time required to achieve convergence, when convergence is specified as reaching 5000 iterations, and the CPU time required by the three stopping rules which intelligently monitor the progress of the calculation metrics (Pop_SD, Elite_Mean and Elite_SD). Although not reported here, the mean precision of the solutions at convergence is very similar. Over the course of repeated calculations and when the dimensionality of the problem increases, the reduced time necessary to achieve convergence is a substantial advantage to the researcher. It is also apparent there is little discernable difference between the CPU time required for convergence when Pop_SD, Elite_Mean and Elite_SD convergence criteria are employed. Potentially, there may be computational advantages to the use of the Elite_Mean approach, however no statistical analysis was undertaken to explore this possibility further.

Comparative Performance

In cases where both approaches are applicable (smooth, continuous twice differentiable functions) evolutionary algorithms have been found to be slower than traditional calculus based approaches. Exploration of the comparative performances of both approaches on the dynamic economic dispatch problem described earlier provides useful context and serves as a point of departure for many of the replicated experiments which will be reported subsequently.

The dynamic economic dispatch problem described earlier in the text and discussed more fully in Appendices 3 and 4 is an example of the type of problem which can be solved by both evolutionary algorithms and using traditional approaches. This problem was strategically constructed expressly to facilitate this performance comparison and the other experiments described here. For purposes

of the replicated performance experiment described here, 50 trials were undertaken on the 24-hour constrained dynamic economic dispatch problem using the summer price vector described in Appendix 15. To facilitate the desired comparison, the population (swarm) size was set at 50 individuals (np=50) for all of the evolutionary algorithms. Certain EA's, such as PSO and RCGA may be disadvantaged by setting the population size to this level for this comparatively small problem. The Elite_SD stopping rule with the elite fraction set to 0.90 was employed for all of the evolutionary algorithms with the convergence tolerance set at 1.0e-04. The Lambda Search algorithm is a deterministic algorithm and, for a given starting point in the search space, the solution results produced are identical for each replication. For this reason the LS algorithm was run only once. For the LS algorithm, the convergence tolerance was set to 1.0e-08. The parameter settings chosen produce a water release of 10,000 af (to two decimal digits of precision) for both the evolutionary algorithms and the Lambda Search algorithm, again facilitating this performance comparison.

Figure 13 compares the convergence behavior of the Lambda Search algorithm with that of the EA's for the first 50 iterations. As shown in this plot, the LS algorithm is initially quite far from the optimal point, and then oscillates around the optimal point rather wildly. This behavior illustrates its relatively poor global search capabilities. Once it identifies the optimal region however, the oscillatory behavior dampens and the algorithm rapidly converges. The LS algorithm achieves convergence to a tolerance of 1.0e-08 in approximately 28 iterations. In contrast, the EA's are able to identify the region containing the optimum point, very quickly, illustrating their relative strength in global search. However, the three EA's require many additional iterations to converge on the optimal point (Refer to Figure 11 for further insights on this subject). Again, this illustrates their relatively poor local search capabilities.

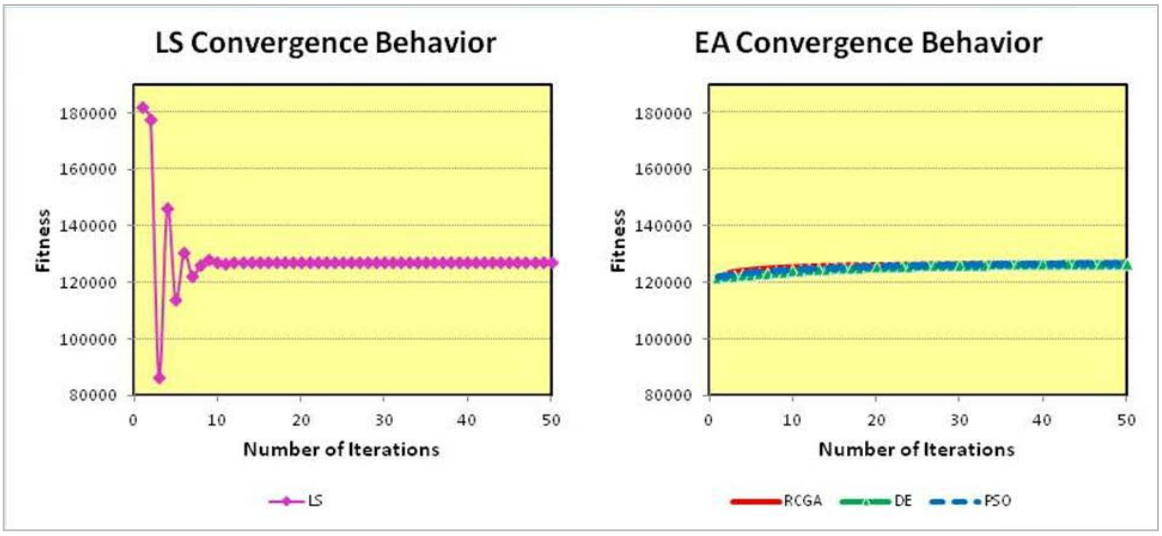


Figure 13. Convergence behavior over 50 Iterations

Table 7 summarizes the convergence results for each of these algorithms and the quality of the solutions they identify. As shown in Table 7 the LS algorithm is able to converge on a final solution very quickly relative to any of the EA's. Over the 50 trials, the EA's are able to identify the same mean solution, albeit at a much higher cost in terms of the number of iterations and the central processing unit (CPU) time required.

Table 7. Convergence Performance and Cost

Algorithm	Mean Best Solution	S.D. Best Solution	Mean Iter	Mean CPU time (msec)
LS[1]	127,097.33	na	28	≤0.002
RCGA	127,097.24	6.198e-01	500	196
DE	127,097.33	1.645e-04	242	114
PSO	127,097.33	1.960e-04	445	500

[1] Lambda search is a deterministic approach and each trial produces the same outcome. The results reported here were generated by a single trial at ctol=1.0e-08.

In some ways, the performance comparison illustrated in Figure 13 and Table 7 is not fully representative of the differences which might be expected in applied work. The Lambda Search algorithm (Appendix 7) is perhaps the fastest of the traditional calculus based methodologies which can be applied to this particular problem. It exploits the structure of this class of constrained optimization problem to reduce the number of decision variables from dim=24 to dim=1 (λ). The LS algorithm is thus a univariate optimization approach and, as such, needs to identify the value of only a single unknown variable, rather than 24 or 168 unknown variables. Naturally, it is quite fast! More general approaches such as the Newton-Raphson approach described in Appendix 6 and the generalized reduced gradient (GRG) method used in spreadsheet solvers, require the addition of artificial variables and slack variables (see Appendix 5). For these algorithms, the total number of unknown variables is typically much larger than the dimensions of the problem. Moreover, commercially available optimization engines typically employ numeric derivatives rather than the analytic derivatives found in the LS algorithm. In aggregate, these requirements increase the problem size (often more than double the problem size) and complexity. These factors may greatly reduce the apparent computational advantages of calculus based approaches demonstrated here.

Problem Dimensions and Input Vectors

At early stages of this research project, an increase in solution times was observed as the size of the constrained optimization problem increased. Similar increases in the solution times are the norm for calculus based optimization approaches. A systematic investigation of this (apparent) performance degradation seemed warranted.

A priori, the influence of increasing the dimensionality of the optimization problem seemed relatively easy to foresee. Increases in the problem size are expected to increase the number of unknown variables, the size of the storage vectors needed, the time to manipulate those vectors and the computational cost of evaluating the fitness function.

The influence of changing the input price vectors is somewhat less obvious. Figure 14 illustrates the 168-hour summer and winter input hourly price vectors used in this analysis and reported in Appendices 16 and 17. This 1-week graph starts on a Sunday, on the left-hand side and ends on a Saturday, on the right-hand side. In the summer, electricity prices are generally higher, have only a single peak during the day and typically have a greater daily range than do the winter prices. The winter prices exhibit the typical two-peaks per day characteristic of electricity prices in cold weather climates.

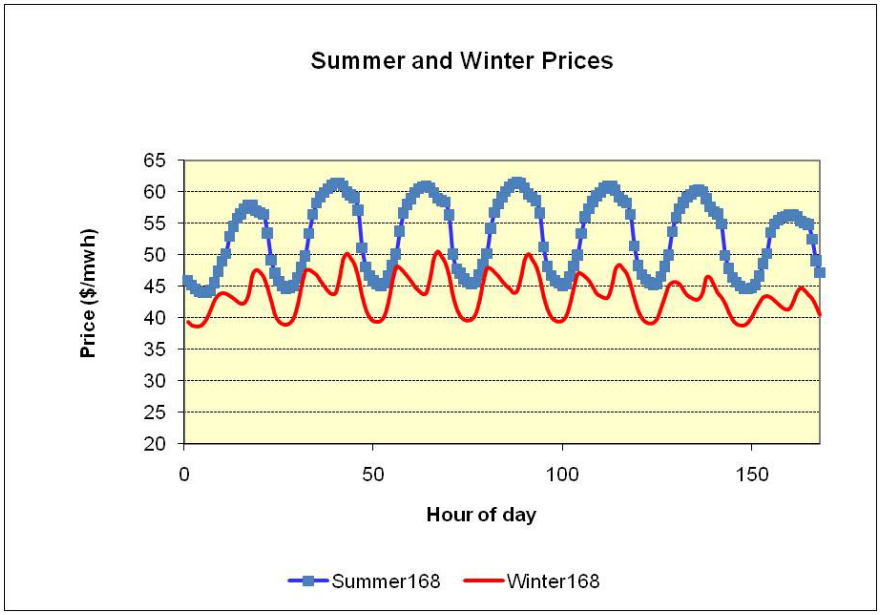


Figure 14. Prices used for analysis.

Examination of Figure 14 might lead to two different hypotheses about the influence of summer and winter input price vectors on convergence speed. The winter prices have more variation in any given day. This might lead to the hypothesis the winter price vector might cause slower convergence. Alternatively, the summer prices have a greater range in magnitude during any given day. This would lead to the hypothesis the summer prices might result in slower convergence speeds. It is unclear which of these two hypotheses might be supported by the experimental outcomes.

Both the potential effect of increased problem size and the use of different input price vectors were explored. For purposes of the replicated stopping rule

experiments, 50 trials were conducted on both the 24-hour and 168-hour constrained dynamic economic dispatch problems using both the summer and winter price vectors. For these experiments, the population size (np) varied across the different algorithms as shown in Tables 3, 4, and 5 found earlier in the document. The Elite_SD stopping rule was employed with the elite fraction set to 0.90. In all cases the convergence tolerance (ctol) was set at 1.0e-04.

The detailed results of this experiment are reported in tabular form in Appendix 18. Figure 15 shown below summarizes the results of this experiment. This figure compares the mean central processing unit (CPU) time, measured in seconds, required for convergence over the 50 trials.

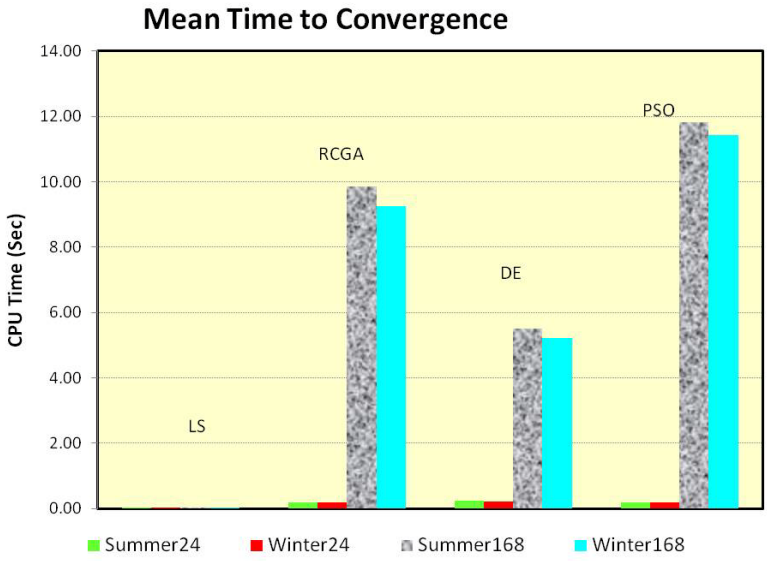


Figure 15. Results of Dimension and Input Vector Experiment

As shown in Figure 15, there is a marked difference in convergence times, across all of the EA's, when the size (dimensions) of the problems are increased from 24 to 168 hours (a 7-fold increase in dimension). The associated increase in convergence time is much greater than proportional to the increase in problem size. The DE algorithm seems to have a clear performance advantage relative to the other EA's tested. This outcome is consistent for all of the trials in which only one constraint (the release volume equality constraint) is binding. However, this performance advantage is not observed when additional constraints are binding, as described in the following section.

Inspection of this plot also shows the choice of input price vector appears to have some influence on convergence times. Convergence times when the summer price input vector is employed are slightly longer than when the winter price vector is used. The statistical significance of this difference, if any, remains to be explored.

Binding Constraints

During the development process, an increase in solution times was observed when there were minimum or maximum binding constraints (other than the total release constraint, which is always binding). This was not necessarily expected by the research team since calculus based optimization approaches do not, apparently at least, suffer as much from this phenomenon. A systematic investigation of this (apparent) performance degradation seemed warranted.

A mixed system of penalty functions and repair methods were employed for all of the EA's examined in this project. Some of the repair approaches are quite involved. More frequent and intensive calls to these penalty and repair subroutines are likely to result in longer solution times.

The potential effects of minimum and maximum binding constraints were systematically explored. Based on the results of previous experiments, it was thought there could be an interaction effect between the binding constraints and the input price vectors employed. A 2x2 experimental design was used to investigate this possibility.

For purposes of the constraint experiments, 50 trials were conducted using the 168-hour constrained dynamic economic dispatch problems. Both the summer and winter price vectors were used in these comparisons. For these experiments, the population size (np) varied across the different algorithms as shown in Tables 3 through 5 found earlier in the text. The Elite_SD stopping rule was employed with the elite fraction set to 0.90. In all experiments the convergence tolerance (ctol) was set at 1.0e-04.

The detailed results of these experiments are reported in tabular form in Appendices 19 and 20. Figures 16 and 17 neatly summarize the results of these experiments. Both figures compare the mean central processing unit (CPU) time, measured in seconds, required for convergence over the 50 trials.

As shown in Figure 16, when the maximum release constraint is set to 6,000 cfs (binding), there is a marked difference in convergence times for all of the EA's examined. Relative to the base case (when the maximum release constraint is not binding), for DE and PSO, the convergence times increase. While for RCGA, the convergence time appears to decrease, relative to the nonbinding maximum release constraint case. Some degradation of the solution quality was apparent, particularly for RCGA and PSO (See Appendices 19 and 20).

Inspection of Figure 16 also suggests that when the maximum release constraint is binding, the choice of input price vector has an independent influence on convergence times. For two of the three EA's (RCGA and DE), convergence times when the summer price input vector is employed are slightly longer than

when the winter price vector is used. For PSO, the opposite appears to be the case. The statistical significance of these differences remains to be explored.

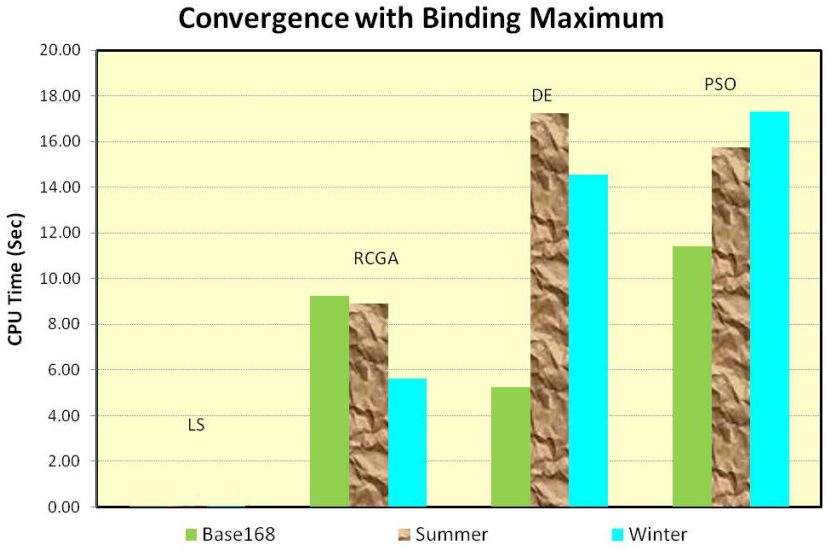


Figure 16. Maximum constraint effects (dim=168).

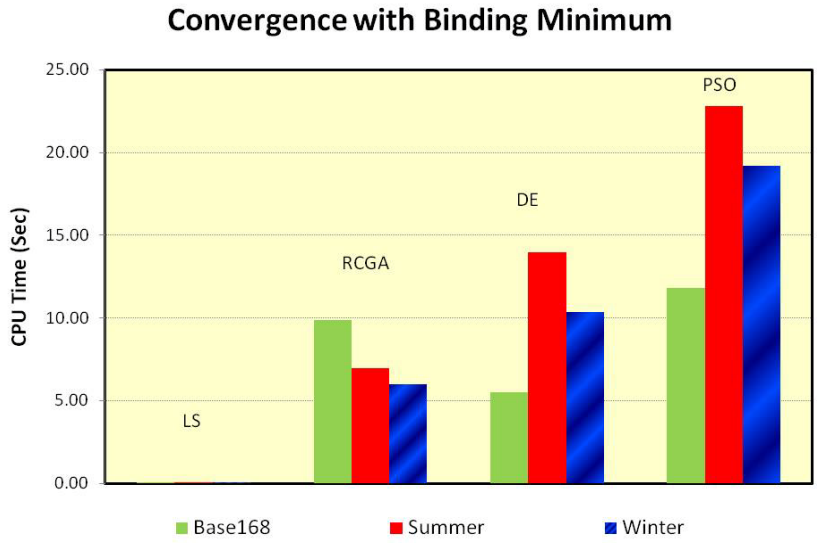


Figure 17. Minimum constraint effects (dim=168).

Figure 17 summarizes the results when the minimum constraint is set to 4,000 cfs (binding). As shown in Figure 17, relative to the nonbinding base case, when the minimum constraint is binding, there are apparent difference in convergence times, across all of the EA's examined. Relative to the nonbinding base case, for DE and PSO, the convergence times increase, while for RCGA, it again appears to decrease.

Further inspection of this plot (Figure 17) also shows that when the minimum release constraint is binding, the choice of input price vector appears to have an independent influence on convergence times. Convergence times for RCGA, DE and PSO are slightly longer when the summer price input vector is employed. Again, the statistical significance of these apparent differences remains to be explored in the future.

To reiterate, the results obtained in these and preceding experiments are relatively voluminous. They include not only the results described here but additional metrics of solution quality and dispersion. Further results from the release constraint experiments can be found in Appendices 19 and 20.

Conclusion

Three promising evolutionary algorithms (EA's) were identified from the emerging heuristic optimization literature; the real coded genetic algorithm (RGGA), differential evolution (DE) and particle swarm optimization (PSO). These algorithms were applied to an important hydropower problem—the constrained dynamic economic dispatch problem. A relatively extensive suite of replicated experiments were conducted to assess their performance characteristics. These experiments systematically explored the influence of initialization approaches, convergence criteria, the dimensions of the problem, the role of problem inputs and the effects of binding constraints. The results show the convergence behavior of evolutionary algorithms differs from traditional calculus based approaches. Evolutionary algorithms exhibit longer solution times—characterized by rapid identification of the region containing the optimum, with relatively slow local convergence. The choice of different initialization approaches appears to have no effect on solution times, for the particular problem examined. Replicated experiments indicate convergence times for all three EAs are longer for higher dimension problems. For DE and PSO, convergence times increase when additional constraints are binding. Input price vectors with greater dynamic ranges appear to degrade convergence times for DE, with mixed results for PSO. The aggregate experimental evidence indicates these algorithms can reliably solve this class of problem, within acceptable time-frames. Many applied hydropower optimization problems are discrete, non-convex and discontinuous. These characteristics preclude the application of traditional calculus-based algorithms. In contrast, evolutionary algorithms are readily applied to such problems and could provide near real-time solutions and guidance for everyday operational decisions at Reclamation's hydropower plants.

Future Directions

The replicated experimental results described here indicate the three selected evolutionary algorithms, PSO, RCGA and DE, are able to accurately and reliably solve the constrained dynamic economic dispatch problem. For optimization problems where both methods are applicable (e.g. smooth, convex functions), evolutionary algorithms have considerably longer solution times than traditional, calculus-based approaches. The strength of evolutionary algorithms however, is they are able to successfully solve a much broader range of problems, including discrete, discontinuous and non-convex problems. The hydropower unit commitment problem is such a problem, and one with widespread, practical, everyday management application at Reclamation hydropower plants.

Research applying these algorithms to the more complex hydropower unit commitment problem is now ongoing. This effort is entitled, *Phase 2- Advanced Optimization Algorithms for Hydropower Dispatch*, Project ID 3906 and is scheduled for completion in Fiscal Year 2013.

Collaborators

Members of the collaborative research team played a pivotal role in the success of this research project. Their generous and invaluable technical contributions to this effort are gratefully acknowledged. The research team for this project was comprised of the following individuals.

Dr. Craig A. Bond, Assistant Professor
Department of Agricultural and Resource Economics
Colorado State University
Ft. Collins, Colorado 80523
(970) 491-6951
Craig.bond@colostate.edu

Dr. Darrell G. Fontane, Professor
Department of Civil Engineering
Colorado State University
Ft. Collins, Colorado 80523
(970) 491-5247
fontane@engr.colostate.edu
Darrell.Fontane@ColoState.Edu

Dr. George W. Heine, Mathematical Analyst
National IRM Center
U.S. Bureau of Land Management
Denver, Colorado 80225
(303) 236-0099
George_heine@blm.gov

Mr. Thomas D. Veselka, Senior Power Engineer
DIS Division
Argonne National Laboratory
Argonne, Illinois 60439
(630) 252-3449
tdveselka@anl.gov

Literature Cited

Ao, Youyun and Hongqin Chi. "Dynamic Differential Evolution for Constrained Real-Parameter Optimization." *Journal of Advances in Information Technology* Vol.1 No. 1. (February 2010):43-51.

Banks, Alec, Jonathan Vincent and Chukwudi Anyakoha. "A Review of Particle Swarm Optimization. Part I: Background and Development. *Natural Computing* Vol 6 No. 4 (December 2007): 467-484.

Banks, Alec, Jonathan Vincent and Chukwudi Anyakoha. "A Review of Particle Swarm Optimization. Part II: Hydridization, Combinatorial, Multicriteria and Constrained Optimization, and Indicative Applications." *Natural Computing* Vol 7 No. 1 (March 2008): 467-484.

Boyd, Stephen and Lieven Vandenberghe. *Convex Optimization*. New York City, New York: Cambridge University Press. 2004. (revised 2006). 730 pages.

Brest, Jamez, Viljem Zumer and Mirjam Sepesy Maucec. "Self-Adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization." Pages 919-926 in, Proceedings of the 2006 IEEE Congress on Evolutionary Computation. Vancouver, British Columbia, Canada. July 16-21, 2006.

Blackwell, Tim Jurgen Branke and Xiaodong Li. "Particle Swarms for Dynamic Optimization Problems." In, *Swarm Intelligence—Introduction and Applications.* Springer Natural Computing Series. Leiden Center for Natural Computing. Christian Blum and Daniel Merkle, Editors. Springer-Verlang: Berlin, Germany. 2008.

Blum, Christian and Xiaodong Li. "Swarm Intelligence in Optimization." In, *Swarm Intelligence—Introduction and Applications.* Springer Natural Computing Series. Leiden Center for Natural Computing. Christian Blum and Daniel Merkle, Editors. Springer-Verlang: Berlin, Germany. 2008.

Boyang Li, Yew-Soon Ong, Minh Nghia Le, Chi Keong Goh: Memetic Gradient Search. IEEE Congress on Evolutionary Computation 2008: 2894-2901.

Bratton, Don and Tim Blackwell. "A Simplified Recombinant PSO." In, *Swarm Intelligence—Introduction and Applications.* Springer Natural Computing Series. Leiden Center for Natural Computing. Christian Blum and Daniel Merkle, Editors. Springer-Verlang: Berlin, Germany. 2008.

Bucknall, Julian. "Ant Colony Optimizations." *The Delphi Magazine* Issue 136 (December 2006):17-22.

Bucknall, Julian. "Round & Round—How Random are Your Numbers?" *The Delphi Magazine* Issue 33 (May 1988):18-25.

Caldwell, Chris. "The Prime Pages—Prime Number Research, Records and Resources." University of Tennessee. http://primes.utm.edu . Last accessed on 12/17/2009.

Carlisle, Anthony and Gerry Dozier. "An Off-The-Shelf PSO." Proceedings of the Workshop on Particle Swarm Optimization. Indianapolis, IN. 2001.

Chakraborty, Uday K (Editor), *Advances in Differential Evolution.* Studies in Computational Intelligence, Volume 143. Springer-Verlang: Belin, Germany. 2008.

Chandrum, K., N. Subrahmanyam and M. Sydulu. "Brent Method for Dynamic Economic Dispatch with Transmission Losses." *Iranian Journal of Electrical and Computer Engineering* Vol. 8 No. 1 (Winter-Spring 2009):16-22,

Coelho, Leandro do Santos and Viviana C. Mariani. "Improved Differential Algorithms for Handling Economic Dispatch Optimization with Generator Constraints." *Energy Conversion and Management.* Vol. 48 No. 5 (May 2007):1631-1639.

Coello Coello, Carlos A. "Theoretical and Numerical Constraint-Handling Techniques Used With Evolutionary Algorithms: A Survey of the State Of the Art." *Computer Methods in Applied Mechanics and Engineering* 191 No. 11-12 (January 2002): 1245-1287.

Computer Dictionary Online. Web based dictionary of computer terms. Located at: www.computer-dictionary-online.org Last accessed on 28 September 2010.

Clerc, Maurice. "Initializations for Particle Swarm Optimization." Unpublished manuscript. 24 December 2008. Available from: http://clerc.maurice.free.fr/pso/. Last accessed on 12/31/2009.

Clerc, Maurice. "Confinements and Biases in Particle Swarm Optimization." Unpublished manuscript. 12 March 2006. Available from: http://clerc.maurice.free.fr/pso/ . Last accessed on 01/04/2010.

Clerc, Maurice. *Particle Swarm Optimization*. London, England: ISTE Publishing Company. 2006.

Clerc, Maurice and James Kennedy. "The Particle Swarm—Explosion, Stability and Convergence in a Multidimensional Complex Space." *IEEE Transactions on Evolutionary Computation* Vol 6 No. 1 (February 2002): 58-73.

Cutello, Vincenzo and Giuseppe Nicosia "An Immunological Approach to Combinatorial Optimization Problems" in, *Lecture Notes in Computer Science*, Vol. 2527. Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence. 2002 pp. 361–370.

Das, Swagatam, Ajith Abraham and Amit Konar. "Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives." *Advances of Computational Intelligence in Industrial Systems* Vol. 116. Ying Liu et al. (Eds.), Studies in Computational Intelligence, Springer Verlag, Germany, 2008. pp. 1–38.

Deb, Kalyanmoy. "An Efficient Constraint Handling Method for Genetic Algorithms." *Computer Methods in Applied Mechanics and Engineering* Vol. 186 No. (2-4) (June 2000): 311-338.

Deep, Kusum and Manoj Thakur. "A New Crossover Operator for Real Coded Genetic Algorithms." *Applied Mathematics and Computation* Vol. 188 No. 1 (May 2007):895-911.

De Jong, Kenneth A. "Analysis of the Behavior of a Class of Genetic Adaptive Systems." Unpublished Ph.D. Dissertation. Computer and Information Sciences. University of Michigan, Ann Arbor. 1975.

Dorigo, Marco and Thomas Stutzle. *Ant Colony Optimization*: MIT Press, Inc. July 2004. 319 pages.

Eberhart, Russell C. and James Kennedy. "A New Optimizer Using Particle Swarm Theory." Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, October 1995; 39-43.

Edwards, Brian K. *The Economics of Hydroelectric Power*. New Horizons in Environmental Economics. Northampton, Massachusetts: Edward Elgar Publishing Inc. 2003.

Edwards, Brian K., Silvio J. Flaim, and Richard E. Howitt. "Optimal Provision of Hydroelectric Power Under Environmental and Regulatory Constraints." *Land Economics*. Vol 75 No. 2 (May 1999):267-283.

Edwards, Brian K., Richard E. Howitt, and Silvio J. Flaim. "Fuel, Crop, and Water Substitution in Irrigated Agriculture." *Resource and Energy Economics* 18 No. 3 (October 1996):311-331.

Engelbrecht, Andries P. *Fundamentals of Computational Swarm Intelligence*. Hoboken, New Jersey: John Wiley & Sons, Ltd. 2005.

Farmani, Raziyeh and Jonathan A. Wright. "Self-Adaptive Fitness Formulation for Constrained Optimization." *IEEE Transactions on Evolutionary Computation* Vol. 7 No. 5 (October 2003):445-455.

Feoktistov, Vitaliy. *Differential Evolution—In Search of Solutions* Volume 5 in Optimization and its Applications Series. Panos M. Pardalos, Managing Editor. Springer, New York, NY. 2006.

Forsund, Finn R. *Hydropower Economics* International Series in Operations Research and Management Science. Frederic S. Hillier, Series Editor. New York, NY. Springer. 2010.

Fylstra, Daniel, Leon Lasdon, John Watson and Allen Warren. "Design and Use of the Microsoft Excel Solver." *Interfaces* 28 No. 5 (September-October) 1998:29-55.

General Electric Energy Corporation. *Western Wind And Solar Integration Study* Prepared for the National Renewable Energy Laboragory. GE Energy. Schenectady, NY. May 2010. 536 pages.

General Electric Energy Corporation. MAPS™ Multi-Area Production Simulation Model Product Description. GE Energy. Schenectady, NY. March 2008. 33 pages.

Goldberg, David E. *Genetic Algorithms in Search Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley Professional Inc. 1989 (reissued). 432 pages.

Gong, Wenyin, Alvaro Fialho and Zhihua Cai. "Adaptive Strategy Selection in Differential Evolution." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2010. July 7-11, 2010. Portland, Oregon. ACM Press, Inc. 2010.

Haupt, Randy L. and Sue Ellen Haupt. *Practical Genetic Algorithms*. 2nd Edition. John Wiley and Son, Inc, New York, N.Y. 2004. 192 pages.

Harpman, David A. "Assessing the Short-Run Economic Cost of Environmental Constraints on Hydropower Operations at Glen Canyon Dam." *Land Economics* 75 No. 3 (August 1999):390-401.

Helwig, Sabine and Rolf Wanka. "Particle Swarm Optimization in High-Dimensional Bounded Search Spaces." Proceedings of the 2007 IEEE Swarm Intelligence Symposium. 1-5 April 2007 Honolulu, HI, Pages 198-205.

Holland, John H. *Adaption in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press. 1975.

Hu, Xiaohui and Russell Eberhart. "Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization." Pages 203-206 in, Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002), Orlando, USA. 2002.

Huang, Hua-Juan and Yong-Quan Zhou. "Hybrid Artificial Fish Swarm Algorithm For Global Optimization Problems" *Journal of Computer Applications*. Vol. 28, no. 12 (December 2008): 3062-3064.

Judd, Kenneth L. *Numerical Methods in Economics*. Second Printing. Cambridge, Massachusetts: MIT Press. 1999.

Karaboga, Dervis and Bahriye Basturk. "A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm." *Journal of Global Optimization* 39 No. 3 (November 2007):459-471.

Kennedy, James and Rui Mendes. "Population Structure and Particle Swarm Performance." Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress. Honolulu, HI , USA. 12 May 2002 - 17 May 2002. pages 1671 – 1676.

Kennedy, James and Russell C. Eberhart. *Swarm Intelligence*. San Francisco, CA: Morgan Kaufmann Academic Press, 2001.

Kim, Dong Hwa, Ajith Abraham and Jae Hoon Cho. "A Hybrid Genetic Algorithm and Bacterial Foraging Approach for Global Optimization." *Information Sciences* Vol 177 No. 18 (September 2007): 3918–3937.

Kirkpatrick, Scott, Charles D. Gelatt, M. P. Vecchi. "Optimization by Simulated Annealing" *Science*, New Series, Vol. 220 No. 4598 (May 1983): 671-680.

Klimasauskas, Casimir C. "Not Knowing Your Random Number Generator Could be Costly: Random Generators-- Why They are Important." *Personal Computer Artificial Intelligence* Vol 16 No. 3 (May/June 2002):52-59.

Knuth, Donald E. *The Art of Computer Programming: Seminumerical Algorithms* Vol 2. 3rd Edition. Massachusetts: Addison-Wesley Press, Inc. 2002.

Kumar, Awadhesh. "Dynamic Economic Dispatch Using Particle Swarm Optimization." Unpublished Masters Thesis. Electrical and Instrumentation Engineering Department, Thapar University, Patiala. June 2009.

Lee, Kwang Y. and Jong-Bac Park. "Application of Particle Swarm Optimization to Economic Dispatch Problem: Advantages and Disadvantages." in, proceedings of the Power Systems Conference and Exposition, 2006. Atlanta, GA. Oct. 29 - Nov. 1 2006. pages 188 – 192.

Li, Boyang, Yew-Soon Ong, Minh Nghia Lee and Chi Keong Goh. "Memetic Gradient Search." *Evolutionary Computation 2008.* Proceedings of the 2008 IEEE World Congress on Computational Intelligence. Hong Kong, China. June 1-6, 2008. pages 2894 – 2901.

Liu, Hui, Zixing Cai and Yong Wang. "Hybridizing Particle Swarm Optimization with Differential Evolution for Constrained Numerical and Engineering Optimization." *Applied Soft Computing* Vol 10 No. 2 (March 2010):629-640.

Matsumoto, Makoto and Takuji Nishimura. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudorandom Number Generator." *Association for Computing Machinery (ACM) Transactions on Modeling and Computer Simulations* Vol 8 No. 1 (January 1998):3-30.

Mezura-Montes, Efren and Blanca Cecilia Lopez-Ramirez. "Comparing Bio-Inspired Algorithms in Constrained Optimization Problems." Pages 662-669 in, Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore. 2007

Mezura-Montes, Efren, Jesus Velazquez-Reyes and Carlos A. Coello Coello. "Modified Differential Evolution for Contrained Optimization" Pages 25-32 in, Proceedings of the 2006 IEEE Congress on Evolutionary Computation. Vancouver, British Columbia, Canada. July 16-21, 2006.

Mezura-Montes, Efren and Jorge Isacc Flores-Mendoza. "Improved Particle Swarm Optimization in Constrained Numerical Search Spaces." in Raymond Chiong (Editor), *Nature-Inspired Algorithms for Optimisation*, pages: 299-332, Springer-Verlag, Studies in Computational Intelligence Series Vol. 193, 2009.

Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd Ed. Springer. 1996. 387 pages.

Michalewicz, Zbigniew and David B. Fogel. *How to Solve it: Modern Heuristics.* 2nd Ed. Springer. 2010. 554 pages.

Mishra, S.K. "Global Optimization by Differential Evolution and Particle Swarm Methods Evaluation on Some Benchmark Functions." MPRA Paper No. 1005, posted 07 November 2007. http://mpra.ub.uni-muenchen.de/1005 .

Miranda, Mario J. and Paul L. Fackler. *Applied Computational Economics and Finance.* Cambridge, Massachusetts: MIT Press. 2006.

Monson, Christopher K. and Kevin D. Seppi. "Linear Equality Constraints and Homomorphous Mappings in PSO." in, Proceedings of the 2005 IEEE World Congress on Evolutionary Computation, Vol. 1 pages 73-80. Edinburgh, Scotland. September 5, 2005.

Nelder, John A. and Roger Mead. "A Simplex Method for Function Minimization" *Computer Journal* 7 No. 4 (January 1965):308-313.

Neri, Ferrante and Ville Tirronen. "Recent Advances in Differential Evolution: A Survey and Experimental Analysis." *Artificial Intelligence Review* 33 Nos. 1-2 (February 2010): 61-106.

Nguyen, Q. H., Yew-Soon Ong, Natalio Krasnogor. "A Study on the Design Issues of Memetic Algorithm." IEEE Congress on Evolutionary Computation 2007: 2390-2397

Noman, Nasimul and Htoshi Iba. "Accelerating Differential Evolution Using Adaptive Local Search." *IEEE Transactions on Evolutionary Computation* Vol. 12 No. 1 (February 2008):107-125.

Omran, Mahamed G.H. "Using Opposition-based Learning with Particle Swarm Optimization and Barebones Differential Evolution." Chapter 23 in, *Particle Swarm Optimization*. Edited by Aleksandar Lazinica. Vienna, Austria: INTECH-Education and Publishing. January 2009. 476 pages.

Omran, Mahamed G.H., Andries P. Engelbrecht and Ayed Salman. "Bare Bones Differential Evolution." *European Journal of Operations Research* Vol 196 No. 1 (July 2009):128-139.

Paquet, Ulrich and Andries P. Engelbrecht. "Particle Swarms for Linearly Constrained Optimization." *Fundamenta Informaticae* Vol. 76 No. 1-2 (March 2007):147-170.

Paquet, Ulrich and Andries P. Engelbrecht. "A New Particle Swarm Optimiser for Linearly Constrained Optimization." in, proceedings of the 2003 IEEE World Congress on Evolutionary Computation Vol. 1 pages 227-233. Canberra, Australia. December 8-13, 2003.

Pant, Millie, Radha Thangaraj, Ved Pal Singh and Alith Abraham. "Particle Swarm Optimization Using Sobol Mutation." Pages 367 to 372 in, The Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology. Nagpur, India. 16-18 July 2008.

Pant, Millie, Radha Thangaraj and Alith Abraham. "Low Discrepancy Initialized Particle Swarm Optimization for Solving Constrained Optimization Problems." *Fundamenta Informaticae* 95 No. 4 (December 2009):1-21.

Pant, Millie, Ved Pal Singh and Alith Abraham. "Differential Evolution using Quadratic Interpolation for Initializing the Population." Pages 375 to 380 in, Proceedings of the 2009 Advance Computing Conference, IACC 2009. IEEE International. Delhi, India. 6-7 March 2009.

Park, Stephen K. and Keith W. Miller, "Random Number Generators: Good Ones Are Hard to Find." *Communications of the Association for Computing Machinery (ACM)*, Vol. 31 No. 10 (October 1988): 1192-1201.

Parsopoulos, Konstantinos E, and Michel N. Vrahatis. "Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method" In, *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*. The Artificial Intelligence Series. Edited by A. Grmela and N.E. Mastorakis. World Scientific and Engineering Academy and Society (WSEAS) Press: Interlaken, Switzerland. 2002.

Pedersen, Magnus Erik Hvass. "Tuning & Simplifying Heuristical Optimization." Unpublished Ph.D. Thesis. School of Engineering Sciences, Computational Engineering and Design Group. University of Southampton. England. January 2010.

Peltokangas Riikka and Aki Sorsa. "Real-coded Genetic Algorithms and Nonlinear Parameter Identification." Report A No. 48. Control Engineering Laboratory, University of Oulu. April 2008.

Pham, D.T., A. Ghanbarzadeh, E. Koc., S. Otri, S. Rahim and M. Zaidi. "The Bees Algorithm—A Novel Tool for Complex Optimization Problems." Innovative Production Machines and Systems 2006 Virtual Conference. July 2006. Available from: http://conference.iproms.org/forums/iproms_2006/optimisation_techniques. Last accessed on 12/29/2009.

Pomeroy, Paul. "An Introduction to Particle Swarm Optimization." March 2003. Online article accessed at: www.adaptiveview.com/articles.

Potter, Walter D., Eric Drucker, Pete Bettinger, Frederick Maier, Max Martin, D. Luper, M. Watkinson, G. Handy, C. Hayes. "Diagnosis, Configuration, Planning, and Pathfinding: Experiments in Nature-Inspired Optimization." *in, Natural Intelligence for Scheduling, Planning and Packing Problems.* Studies in Computational Intelligence. Vol. 250. Berlin, Germany: Springer Verlag. 2009. pages 267-294.

Press, William H., Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling. *Numerical Recipes in Pascal—The Art of Scientific Computing.* New York: Cambridge University Press. 1989.

Price, Kenneth V. and Rainer M. Storn. "Differential Evolution" *Dr. Dobb's Journal* Issue 264 (April 1997):18-24 and 78.

Price, Kenneth V., Rainer M. Storn and Jouni A. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization.* Springer-Verlang: Belin, Germany. 2005. 538 pages.

Rahnamayan, Shahryar and G. Gary Wang. "Solving Large Scale Optimization Problems by Opposition-Based Differential Evolution (ODE). "*WSEAS Transactions on Computers* 10 Vol. 7 (October 2008):1792-1804.

Rahnamayan, Shahryar, Hamid R. Tizhoosh and Magdy M.A. Salama. "Opposition-Based Differential Evolution." Chapter 6 in, *Advances in Differential Evolution.* Studies in Computational Intelligence, Volume 143. Edited by Uday K. Chakraborty. Springer-Verlang: Belin, Germany. 2008.

Rajkumar, N. Timo Vekara and Jarmo T. Alander. "A Review of Genetic Algorithms in Power Engineering." in, *AI and Machine Consciousness—Proceedings of the 13th Finish Artificial Intelligence Conference.* Tapani Raiko, Penti Haikonen and Jaakko Vayrynen, editors. Esppo, Finland. August 20-22, 2008. pages 15-32.

Reeves, Collin R. "Genetic Algorithms." Chapter 3 in, *Handbook of Metaheuristics* 2nd Edition. edited by Michel Gendreau and Jean-Yves Potvin. Springer Business and Science: New York City, NY. 2010. 650 pages.

Rau, Narayan S. *Optimization Principles—Practical Applications to the Operation and Markets of the Electric Power Industry.* IEEE Press Series on Power Engineering. P.M. Anderson, Series Editor. New York, New York: John Wiley and Sons. 2003. 339 pages.

Richards, Mark and Dan Ventura. "Choosing a Starting Configuration for Particle Swarm Optimization." pages 2309–2312 in, *Proceedings of the Joint Conference on Neural Networks.* July 2004.

Robertson, Grant. "How Powerful was the Apollo 11 Computer?" *Download Squad*. Weblogs, Inc. RSS Feed. July 20, 2009 at 8:30 pm.

Shah-Hosseini, Hamedi, "The Intelligent Water Drops Algorithm: A Nature-Inspired Swarm-Based Optimization Algorithm." *International Journal of Bio-Inspired Computation,* Vol. 1, Nos. 1 and 2 (January 2009):71-79.

Simopoulos, Dimitris N. Stavroula D. Kavatza and Costas D. Voumas. "An Enhanced Peak Shaving Method for Short Term Hydrothermal Scheduling." *Energy Conversion and Management* Vol. 40 No. 1 (November 2007): 2018-3024.

Staschus, Konstantin, Andrew M. Bell, and Eileen Cashman. "Usable Hydro Capacity, Electric Utility Production Simulations, and Reliability Calculations." Institute of Electrical and Electronics Engineers (IEEE), *Transactions on Power Systems* Vol 5 No. 2 (May 1990):531-538.

Storn, Rainer M. and Kenneth V. Price. *Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces.* Technical Report TR-95-12, International Computer Science Institute. March 1995. Available from: http://www.icsi.berkeley.edu .

Storn, Rainer M. and Kenneth V. Price. "Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces." *Journal of Global Optimization* 11 No. 4 (December 1997):341-359.

Tang, Jun and Xiaojuan Zhao. "A Hybrid Particle Swarm Optimization with Adaptive Local Search." *Journal of Networks* Vol. 5 No. 4 (April 2010):411-418.

Tang, K., X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. *Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization*. Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007.

Uy, Nguyen Quang, Nguyen Xuan Hoai, RI McKay, and Pham Minh Tuan. "Initializing PSO with Randomised Low-Discrepancy Sequences: The Comparative Results." Pages 1985-1992 in, Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore. 2007.

Valle, Yamille del, Ganesh Kumar Venayagamoorthy, Salman Mohagheghi, Jean-Carlos Hernandez and Ronald G. Harley. "Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems." *IEEE Transactions on Evolutionary Computation* Vol 12 No. 2 (April 2008):171-195.

Veselka, Thomas D., Leslie D. Poch, Clayton S. Palmer, Samuel Loftin and Brent Osiek *Financial Analysis of Experimental Releases Conducted at Glen Canyon Dam during Water Years 1997 through 2005* ANL/DIS-10-7 August 2010. [REVISED VERSION]

Veselka, Thomas D., Leslie D. Poch, Clayton S. Palmer, Samuel Loftin and Brent Osiek. *Ex Post Power Economic Analysis of Record of Decision Operational Restrictions at Glen Canyon Dam* ANL/DIS-10-6. Argonne National Laboratory. Argonne, Illinois. April 2010. 96 pages.

Veselka, Thomas D., O. Benjamin Schoepfle and Matthew Mahalik. *CRSP Basin-Wide Economic Methodology—Modeling the Aspinall Cascade*. Systems Science Group, Argonne National Laboratory. Argonne, Illinois: Argonne National Laboratory. July 2003.

Vicaria, Fernando. "Totally Random—Getting a Truly Random Number." *Delphi Informant* 9. No. 10 (October 2003):8-14.

Wahde, Mattias. *Biologically Inspired Optimization Methods-- An Introduction* Southampton, MA: WIT Press. 2008. 225 pages.

Weber, Ernst Juerg. "Optimal Control Theory for Undergraduates Using the Microsoft Excel Solver Tool." *Computers in Higher Education Economics Review (Cheer)* 19 No. 1 (2007):4-15.

Weise, Thomas. *Global Optimization—Theory and Practice* 2nd Edition. eBook. Version 2008-07-07. Available from: www.it-weise.de. 2008. 703 pages.

Wong, Tien-Tsin, Wai-Shing Luk and Pheng-Ann Heng. "Sampling with Hammersley and Halton Points." *Journal of Graphics Tools* 2 No. 2 (November 1997): 9-24.

Wolpert, David H. and William G. Macready. "No Free Lunch Theorems for Optimization" *IEEE Transactions on Evolutionary Computation* Vol. 1 No. 1 (April 1997):67-82.

Wood, Allen J. and Bruce F. Wollenberg. *Power Generation, Operation and Control.* 2nd Edition. New York, New York: John Wiley and Sons, 1996.

Wright, Alden H. "Genetic Algorithms for Real Parameter Optimization." In, *Foundations of Genetic Algorithms.* Gregory J.E. Rawlins (editor). San Mateo, CA: Morgan Kaufmann. 1991. pp. 205-218.

Yang, Xin-She. "Firefly Algorithm, Stochastic Test Functions and Design Optimization." *International Journal of Bio-Inspired Computation* Vol 2 No. 2 (March 2010):78-84.

Yang, Xin-She and Suash Deb. "Engineering Optimization by Cuckoo Search." *International Journal of Mathematical Modeling and Numerical Optimization.* Vol. 1 No.4 (April 2010):330-343.

Yang, Xin-She and Suash Deb. "Cuckoo Search via Levy Flights." in, *Proceedings of the World Congress on Nature and Biologically Inspired Computing* (NaBIC 2009, India). IEEE Publications, USA. pages 210-214.

Zielinski, Karin and Rainer Laur. "Stopping Criteria for Differential Evolution in Constrained Single-Objective Optimizations." Chapter 5 in, *Advances in Differential Evolution.* Studies in Computational Intelligence, Volume 143. Edited by Uday K. Chakraborty. Springer-Verlang: Belin, Germany. 2008.

Zielinski, Karin, Petra Weitkemper, Rainer Laur and Karl-Dirk Kammeyer. "Examination of Stopping Criteria for Differential Evolution based on a Power Allocation Problem." In, volume 3, pages 149–156, Proceedings of the 10th

International Conference on Optimization of Electrical and Electronic Equipment. Brasov, Romania, 18-19 May 2006.

Zielinski, Karin and Rainer Laur. "Stopping Criteria for Constrained Single-Objective Particle Swarm Algorithm." *Informatica* 31 No. 1 (March 2007):51-59.

Appendix 1. Objectives for Dispatch

Introduction

This appendix compares and contrasts several alternative objective functions which could be used to characterize the dispatch of hydropower plants. In the dynamic economic dispatch problem, the objective function describes the owner/operators' hourly dispatch decisions. This mathematical description of the decision process is crucially important to the outcome of the optimization process. In the vast majority of published literature, it is presumed that hydropower owners dispatch their plants to maximize economic benefits. While this is quite often a prime consideration, it is not the only factor shaping the hourly dispatch decision. In fact, there are a wide range of potential specifications for the objective function. These alternate characterizations may embody a number of real-world strategic and institutional considerations--beyond purely economic motives.

Economic Dispatch

Hydropower dispatch for maximum economic benefit is also known as (pure) economic dispatch. Most models of hydropower operations assume that hydropower plant owners behave as though they have this sole objective and make their dispatch decisions consistent with it. Operationally, the owner/operator of the hydropower plant uses their available storage water to generate electricity when the price is highest, subject to the physical, operational and environmental constraints on the powerplant. Economists are particularly fond of this characterization since it reflects the tenants of economic theory. Many engineering texts also base their expositions on this objective function (e.g. Wood and Wollenberg 1996, Rau 2003).

This objective function represents the maximum economic benefits which could be received in the absence of institutional or other strategic considerations or constraints. Since these institutions or strategic constraints are typically rather dynamic and management specific, they can change at any time. As a result, this objective function may be preferred for long-term economic studies.

Peak Shaving

Typically, the highest observed prices for electricity correspond to the hours when load is greatest. This occurs because the most expensive thermal plants must be dispatched to meet load during these peak periods. Hydropower plants can be employed to reduce the load during these peak periods, reducing the need for

dispatch of expensive thermal plants and reducing the overall system costs. This dispatch strategy is known as, "peak shaving." Peak shaving is the objective function used in a number of electricity system models including General Electric Corporation's well known Multi-Area Production Simulation (MAPS) model (General Electric Corporation 2008). The underlying algorithm was described by Stachus, Bell and Cashman (1990) and later by Simopoulos, Kavatza and Voumas (2007). It has been employed for modeling the complex hydropower operations at Glen Canyon Dam (Harpman 1999). Recently, the peak shaving approach has been used in a variety of renewable resource integration studies including a recent study for the National Renewable Energy Laboratory (General Electric Corporation 2010). In the context of renewable energy studies, the peakshaving algorithm is typically employed to shave the peaks of the load minus the generation from renewable intermittent sources.

Native Load First

The stated objective of some entities is to a serve their native load first, using their owned hydropower resources, and then to profit from the sale of any remaining generation. If they have surplus energy or capacity, after meeting their own system load, they will sell it to others on either the firm or spot markets. If they are resource short, they must buy energy to make up the shortfall. The strategy of serving native load first, may or may not be consistent (probably is not) with either minimizing production costs in the larger interconnected system or maximizing the economic benefits afforded by their owned resources. This objective has been espoused by Western Area Power Administration as describing their operation of the Colorado River Storage Project hydropower resources (Veselka, et al 2010).

Buy/Sell and Generate

A number of larger utilities and other entities have sufficient owned resources and a large enough market presence and access to transmission that they can purchase, sell or generate electricity, at their option. By judiciously purchasing on the open-market, generating when necessary and economic, and selling when electricity prices are high, these entities can leverage the use of their owned hydropower resources and their market presence in order to maximize economic benefits. Such a strategy can be used to increase the benefits afforded by limited pondage (storage) hydropower plants. This strategy is discussed in considerable detail in Edwards, Howitt and Flaim (1996), Edwards, Flaim and Howitt (1999) and Edwards (2003) in association with the optimal operation of hydropower plants in the Upper Colorado River Basin.

Appendix 2. Ancillary Services and Dispatch

Ancillary Services

In addition to providing energy and capacity, hydropower plants are large-scale providers of ancillary services. Ancillary services are defined by FERC (1995) as, "… those services necessary to support the transmission of electric power from the producer to the purchaser given the obligations of control areas and transmitting utilities within those control areas to maintain reliable operations of the interconnected transmission system."

Ancillary services help maintain reliable system operations in accordance with good utility practice. Some of these services include spinning reserve, non-spinning reserve, replacement reserve, regulation/load following, black start, reactive power and voltage support. Quick start times, fast ramping capabilities, and the ability for rapid corrective responses to changes in grid conditions make hydropower plants an excellent resource for providing ancillary services.

Dispatch Effects

Two ancillary services in particular, spinning reserves and regulation, can affect the optimal economic dispatch problem. Figure 18 shows how providing ancillary services can reduce the operating range of a power plant. Spinning reserves reduce maximum scheduled operations. On the other hand, regulation affects both maximum and minimum production levels. The influence of these services on dispatch operations is described below.

Regulation is the amount of operating reserve capacity required by the control area to respond to automatic generation control to assure that the Area Control Error meets these two conditions: that it (1) equals zero at least one time in all 10-minute periods and (2) falls within specified limits to manage the inadvertent flow of energy between control areas.

Hydropower plants can provide regulation services by responding quickly to moment-by-moment up and down movements in control area electricity demand using Automatic Generation Control (AGC). Larger plants, multi-unit plant, are particularly well suited for providing this service because at least one or more of their turbines are always on-line, and they operate at sufficiently high output levels such that sudden decreases in load will not reduce generation below their technical or regulatory minimum output levels.

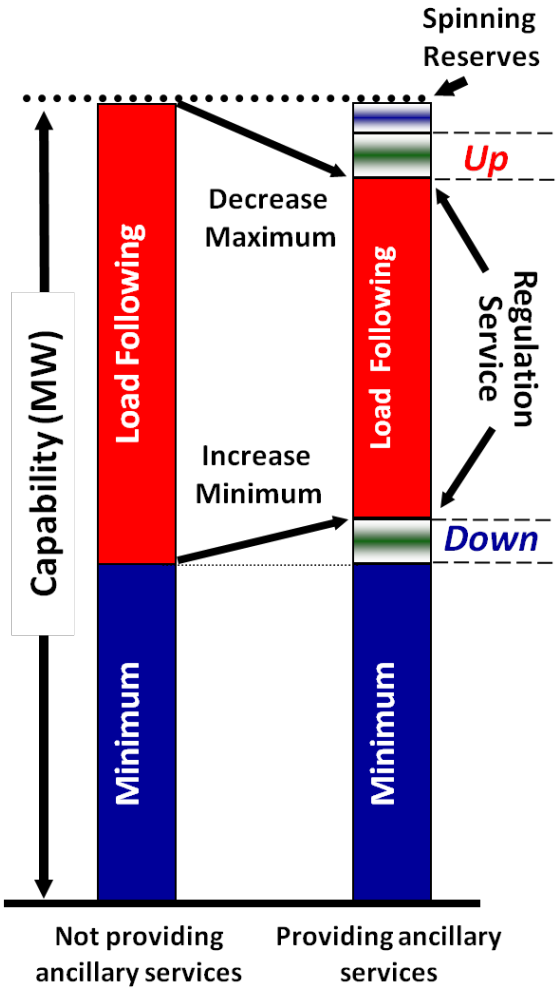


Figure 18. Ancillary services and dispatch.

To provide regulation-up service, generation levels must be sufficiently low such that a power plant can respond to instantaneous decreases in grid loads without exceeding their output capability. Regulation-up services will incur an opportunity cost when maximum power plant sales during peak periods are required to be lower than the plant's capability. The power plant's average hourly production level must be at or below the plant's capability minus the regulation-up service level. Under either scenario, regulation-up service does not incur any opportunity costs under all but very high hydropower conditions since the dam is operating below the maximum power plant capacity. It is of note that at many times, the regulatory flow rate is significantly below the physical plant limit. It should also be noted that providing regulation services may not affect either hourly ramping or daily changes at a powerplant.

Spinning reserves are defined as generating capacity that is running at a zero load, connected to an output bus, synchronized to the electric system, and ready to take immediate load. The portion of unloaded synchronized generating capacity, controlled by the power system operator, must be capable both of being loaded in 10 minutes and kept running continuously for a set period of hours.

When a generator supplies spinning reserve services, it will increase output in response to an outage situation. The increased output fills the generation void created by a generator in a balancing authority that suddenly ceases to produce power. Spinning reserves may also be called upon when an abrupt transmission line outage will no longer permit the reliable transport of power into a region. Generation levels in normal conditions must be sufficiently low such that when an outage occurs, it can increase output levels by its spinning reserve obligation without exceeding the maximum capability of the generator.

Spinning reserve services require that maximum production levels do not exceed the plant's capability minus the amount of spinning reserves required. Providing spinning reserves also requires that one or more turbines operate below capability or in a spinning state without producing power. The former condition may require the unit to operate in a sub-optimal state, while the latter releases water without power production to spin the turbines under no load. These additional requirements typically incur opportunity costs, because capacity must be reserved at the high end of operations to accommodate the spinning reserves. Unlike regulation-down services, spinning reserves do not affect minimum generation levels.

Appendix 3. Hydropower Plant Specifications

Release, Head and Generation

For the dynamic economic dispatch problem, the equation used to characterize real power generation at the hydropower plant is shown in (21).

$$p_i = \frac{q_i \times \gamma \times eff_i \times head(Q, elev)}{fptokw \times 1000} \tag{21}$$

Where:

p_i = (real) electric power generated (mw) by the generator
γ = 62.40, specific weight of water at 50 degrees Fahrenheit (lbs/ft^3).
eff_i= efficiency factor[1] (dimensionless) for the generator.
q_i = release (cfs) from the generator
Q = total release from all sources.
elev = reservoir elevation (ft above mean sea level).
head = net head (ft)
fptokw = 737.5, foot-pounds to kilowatt conversion factor (kW/(ft-lbs/sec)

For purpose of this problem net head is defined as the difference between the reservoir elevation and the tailwater elevation. While it is assumed the reservoir elevation is known, the elevation of the tailwater varies with the total release from all sources, Q. More explicitly, the head depends not only on the releases made from the powerplant but also from the outlet works, if any. For our purposes, we characterize net head as shown in (22).

$$head = [elev - (w0 + w1 \times Q)] \tag{22}$$

Where:

head = generation head (ft)
elev = reservoir elevation (ft above mean sea level).
w0 = 1708.186, tailwater height (ft above mean sea level) when Q=0.0
w1 = 0.0070, change in tailwater elevation as release changes (ft/cfs)
Q = total release from all sources

[1] In this application the efficiency (eff) is represented as a constant. More generally, efficiency may vary as a function of release and head.

The fully specifie relationship for generation at time (t) as a function of release and head is then (23).

(23) $$p_t = \frac{q_t \times \gamma \times eff \times [elev_t - (W0 + W1 \times q_t)]}{fptokw \times 1000}$$

Collecting terms, the generation equation can be further simplified as shown in (24). Note that equation (24) is nonlinear and quadratic in q_t.

(24) $$p_t = \frac{\gamma \times eff \times [(elev_t - W0)q_t + W1q_t^2)]}{fptokw \times 1000}$$

Minimum and Maximum Release Constraints

For the hydropower plant, the minimum and maximum release constraint values are shown in Table 8.

Table 8. Maximum and Minimum Release Constraints

Outlet	Minimum release (cfs)	Maximum release (cfs)
	k	w
Generator	0.0	12,000
Outlet works	0.0	15,000

Generator Specifications

The engineering specifications for the (aggregate) generator characterized at the hydropower plant are shown in Table 9.

Table 9. Generator Specifications

Efficiency (eff)	0.85
Capacity	239.9551 MW[1]

[1] The maximum occurs at an elevation of 2008.18560 and a release of 12,000 cfs.

Plant Description

As described, the relationship between release, net head and generation is relatively complex. Figure 19 illustrates this relationship. At a given reservoir elevation, for releases between the minimum necessary release and the maximum release, the relationship is approximately linear. Once the maximum release capability of a given turbine has been reached, no additional generation from that unit can be achieved. Releases in excess of a given turbine's release capability must be made from another turbine, or from the outlet works. Such releases cause an increase in the tailwater elevation, a decrease in net head and a decline in the generation from that unit.

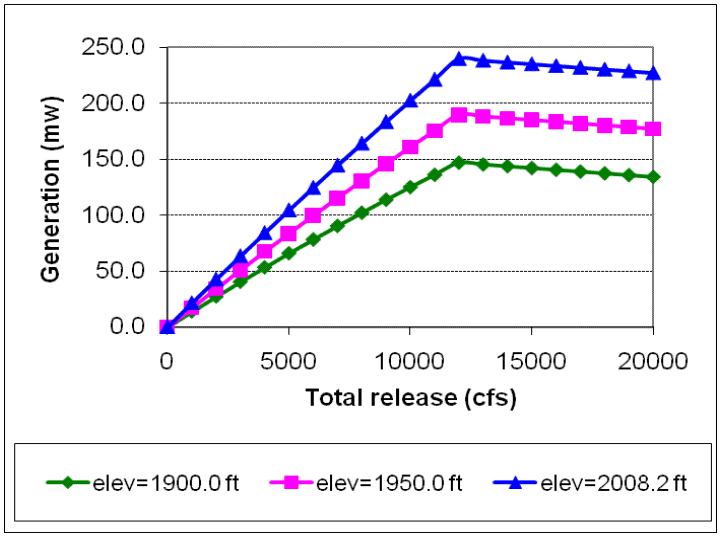


Figure 19. Release, head, and generation relationship.

Appendix 4. Release, Head and Efficiency

The efficiency parameter (eff) for the universal power equation described previously in Appendix 3, determines the rate at which falling water is converted into electrical energy. Efficiency is typically measured as a decimal fraction or a percent. The relationship between release, head and generation contained in Appendix 3 utilizes a static value for the efficiency (eff) which is constant for all values of head and release. In general however, the efficiency of a Francis turbine varies depending on the head and the release rate and this relationship is unique to the design of each turbine runner and the site where it is installed.

In response to previous review comments, this appendix describes the more general relationship between efficiency, release and head. The generic mathematical relationships developed in this appendix are purposely specified in terms of percent of maximum release and percent of maximum head, to allow for the ease of application in this and future research efforts.

To accommodate generic and nonspecific use, the relationship between total release, head and efficiency becomes slightly more complex, but remains reasonably tractable. A plausible relationship between release and head is the quadratic function shown in equation (25).

(25) $$E = \frac{-(q - bestQ)^2}{head} + bestE$$

Where: E = efficiency (dimensionless)
q = total release (cfs)
bestQ = the release yielding the highest value of E
head = gross head (feet)
bestE = the highest value of E which can be attained.

The values for bestQ and bestE for a particular research application must be calculated. The maximum value of E which can be obtained at a given head is computed using expression (26)..

(26) $$bestE = \left(A * \frac{head}{maxhead} + A \right) * 100$$

Where: bestE = the maximum efficiency (dimensionless)
A = scalar parameter ($0<A\leq1$).
head = gross head (feet)
maxhead = the maximum normal gross head at this site.

The value of Q which produces the maximum efficiency, for a given head is described by equation (27).

(27) $$bestQ = bestE + B * (MaxE - bestE)$$

Where: bestQ = the release which produces the maximum value of E (cfs)
bestE = the maximum efficiency (dimensionless)
B = scalar parameter ($0<B\leq1$).
maxE = the value of bestE obtained by evaluating equation (26) with the head set equal to the maximum head dimensionless).

For purposes of this exposition, the values of the parameters used are illustrated in Table 10 shown below. Naturally, these parameter values will vary, depending on the details of the specific research application being examined.

Table 10. Efficiency Parameter Values

Parameter	Value
maxhead	400.00
A	0.450
B	0.400

Using the parameter values shown in Table 10 in expressions 25 and 26, the relationship between release, head and efficiency, described by equation (27) can be plotted for three different levels of gross head as shown in Figure 20.

As illustrated in this figure, the expression for efficiency as a function of release and head (equation 27) provides a very reasonable representation of the relationships between these variables. For instance, at a gross head of 400 feet, the maximum efficiency is 90 percent at a gate opening of 90 percent. At a lower head of 300 feet, the maximum efficiency is 78.75 percent at a gate opening of 83 percent. This relationship closely tracks and is similar to the observed efficiency characteristics at many hydropower facilities where Francis turbines are employed.

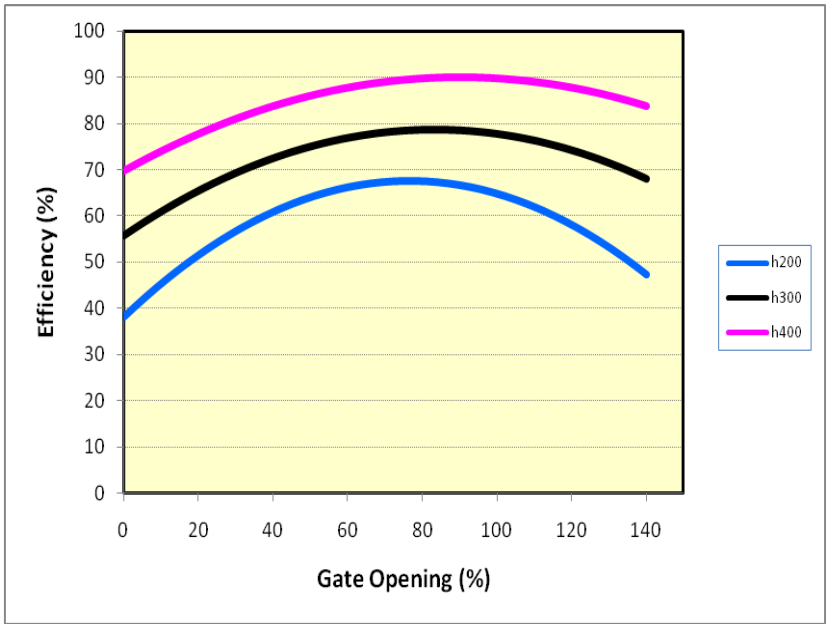


Figure 20. Release, head, and efficiency.

Appendix 5. Calculus of Dynamic Dispatch

This appendix reviews the traditional or calculus based approach for the solution of the optimal dynamic dispatch problem using a specific example. This approach dates back to the days of Sir Isaac Newton and is routinely taught in economics, engineering and physical science classes. To keep the size of the example reasonably tractable, the problem explored here is limited to a two period (T=2) problem. To facilitate illustration, this example problem includes fewer constraints than the general problem.

Example Problem

For purposes of this example, the hydropower plant operator is given an amount of water for release (Q) over a planning horizon of T=2. We assume s/he knows price of electricity (R) over each of the T-periods. The plant operator must decide how much water to release for generation in each period (t) to maximize revenue over the planning horizon. In this example we will presume there are constraints on the total amount of water available for release (Q) and the maximum amount of water which can be released in each period.

This example optimal dynamic dispatch problem can be written in mathematical notation as shown in equations (28) through (30).

(28) $$\textit{Maximize} \quad \sum_{1}^{T} R_t \, p_t(q_t)$$

subject to:

(29) $$\sum_{1}^{T} q_t \le Q$$

(30) $$q_t \le q_{\max} \quad \forall_t \in \{1..T\}$$

Where: R_t = price ($/MWh) at time (t)
p_t = generation (MW) at time (t)
q_t = release (cfs or af) at time (t)
Q = total release (af).
q_{max} = maximum release
T = 2

We will assume here that the maximum release constraints do not vary across the planning horizon. For this reason, q_{max} is the same for all periods (t).

The objective of the plant operator is to maximize revenue over the time horizon (T=2) by generating electricity when it is most valuable. While doing so, s/he cannot exceed the amount of water available for release over the planning horizon (constraint equation 29). The amount of water released in any one period must be less than or equal to the maximum water release constraint (equation 30).

A Specific Example Problem

As described so far, the example is very general. To facilitate further understanding and demonstrate how to solve these types of problems, we will fully specify the mathematical form. Drawing on our previous exposition and Appendix 3, the relationship between generation, release and head can be specified as (31).

(31) $$p_t = \frac{\gamma \times eff \times [(elev_t - W0)q_t + W1q_t^2)]}{fptokw \times 1000}$$

where: γ = 62.40, specific weight of water at 50 degrees Fahrenheit (lbs/ft^3).
eff_i= efficiency factor[2] (dimensionless) for the generator.
elev = reservoir elevation (ft above mean sea level).
fptokw = 737.5, foot-pounds to kilowatt conversion factor (kW/(ft-lbs/sec)
w0 = 1708.186, height of the tailwater (ft above mean sea level) when Q=0.0
w1 = 0.0070, rate of change in tailwater elevation as release changes (ft/cfs)

Exploiting the many constants in (31), it will be expedient to form a simpler constant as shown in (32).

(32) $$Let \quad A = \frac{\gamma \times eff}{fptokw \times 1000}$$

We can then write out the relationship between generation, elevation and release as the simpler and more streamlined expression (33).

(33) $$p_t = A \times [(elev_t - W0)q_t + W1q_t^2]$$

[2] In this application the efficiency (eff) is represented as a constant. More generally, efficiency may vary as a function of release and head.

The fully specified optimal dynamic dispatch problem can be written in mathematical notation as shown in equations (34) through (36).

(34) $$Maximize \quad \sum_{1}^{T} R_t \times A \times [(elev_t - W0)q_t + W1q_t^2]$$

subject to:

(35) $$\sum_{1}^{T} q_t \le Q$$

(36) $$q_t \le q_{\max} \quad \forall_t \in \{1..T\}$$

This class of constrained optimization problem occurs frequently and conceptually can be solved using the Kuhn-Tucker conditions or by the introduction of artificial slack variables. We choose the latter approach, since arguably it is both simpler and more tractable.

Introducing Slack Variables

The constraints (35 to 36) in the fully specified example problem above are inequality equations. We can convert these inequalities to equalities by introducing some artificial variables known as "slack variables." We will denote the slack variables as, "S_x" where the subscript (x) identifies the context of the specific slack variable. For purposes of the numerical solution example (introduced subsequently), we will insure the slack values in each equation are nonnegative, by specifying them as squared terms. This serves to restrict the value of the (squared) slack variable to be zero, or some positive value.

Introducing slack variables into the inequality constraints (35 to 36) yields the corresponding equality constraints (37 to 38).

(37) $$\sum_{1}^{T} q_t + S_Q^2 = Q$$

(38) $$q_t + S_{\max}^2 = q_{\max}$$

A closer look at the water balance equation (37) will help to illustrate the use of slack variables and their meaning in an equation. In the event less water is released over the planning horizon than is available (the sum of the q_t is less than Q), the value of S^2 will be positive and nonzero. This indicates there is extra or "slack" water remaining in the reservoir which has not been released for generation. If the constraint is binding (the sum of the amount of water release exactly equals Q), then the value of S^2 will be zero. This indicates that all of the

water available for release during the planning horizon has been released and there is no extra water (no slack) remaining.

Slack Variable Formulation

Incorporating the constraint equations as equality constraints with slack variables, the fully specified example problem then can be restated as equations (39) through (41).

(39) $$Maximize \quad \sum_{1}^{T} R_t \times A \times [(elev_t - W0)q_t + W1q_t^2]$$

subject to:

(40) $$\sum_{1}^{T} q_t + S_Q^2 = Q$$

(41) $$q_t + S_{\max,t}^2 = q_{\max} \quad \forall_t \in \{1..T\}$$

An analytical solution to this problem can be found using the method of Lagrange. We do so by first forming a Lagrangian expression and then maximizing it as shown in (42).

(42) $$Max\ L = \sum_{1}^{T} R_t \times A \times [(elev_t - W0)q_t + W1q_t^2] + \lambda(Q - S_\lambda^2 - \sum_{1}^{T} q_t) + \sum_{1}^{T} \gamma_t (q_{\max} - q_t + S_{\gamma,t}^2)$$

A Lagrangian expression embodies both the original objective function, and each of the constraints. The introduction of Lagrange multipliers (λ, γ) is a clever mathematical device. The underlying logic is that either the value of the Lagrange multiplier is zero, the value in parentheses (the constraint) is zero, or both. The resulting Lagrangian expression incorporates T*1+1 or 3 new Lagrangian variables; one Lambda (λ) for the water release constraint and a set of T Gammas (γ), one for each maximum release constraint.

There are 8 choice variables in the Lagrangian expression (equation 42). The set of choice variables includes T=2 water releases (the q's), the (T*1+1) or 3 Lagrangian variables and the (T*1+1) or 3 slack variables (S).

Analytic Solution

The first order necessary conditions (FOC's) for a maximum require the first derivatives of the function be equal to zero at the optimum. Taking the first partial derivatives of the Lagrangian function (42) with respect to each of the 8 choice variables, yields the set or vector of 8 differential equations shown in equations (43) to (50).

(43) $$\frac{\partial L}{\partial q_1} = -R_1 AW0 + 2R_1 AW1 q_1 - \lambda - \gamma_1 = 0$$

(44) $$\frac{\partial L}{\partial q_2} = -R_2 AW0 + 2R_2 AW1 q_2 - \lambda - \gamma_2 = 0$$

(45) $$\frac{\partial L}{\partial \lambda} = (Q - S_\lambda^2 - \sum_1^T q_t) = 0$$

(46) $$\frac{\partial L}{\partial \gamma_1} = (q_{\max} - q_1 + S_{\gamma,1}^2) = 0$$

(47) $$\frac{\partial L}{\partial \gamma_2} = (q_{\max} - q_2 + S_{\gamma,2}^2) = 0$$

(48) $$\frac{\partial L}{\partial S_\lambda} = -2\lambda S_\lambda = 0$$

(49) $$\frac{\partial L}{\partial S_{\gamma,1}} = 2\gamma_1 S_{\gamma,1} = 0$$

(50) $$\frac{\partial L}{\partial S_{\gamma,2}} = 2\gamma_2 S_{\gamma,2} = 0$$

Conceptually, this set of simultaneous differential equations can be solved to find the values of q, λ, γ and S. And for this relatively simple example, analytic solution is certainly feasible. In many practical applications, T is larger and there are many more constraints. In many, if not most real-world applications, it is not possible to solve this type of problem analytically.

Lagrange Multipliers

Lagrange multipliers hold a special significance to economists, engineers and economists. They are interpreted as the shadow price, dual price or the value of the marginal unit of the resource. To place this in context, consider Lambda (λ), the Lagrange multiplier for the water constraint. Lambda represents the marginal value of an additional unit of water when allocated optimally. When Lambda is positive, this indicates that an additional unit of water will have a value of λ. If Lambda is zero, this suggests that an additional unit of water will not add any value, if more were available. When Lambda is negative, this indicates that additional releases will cause the maximum revenue to decline.

Appendix 6. Newton-Raphson Method

Newton's method, or the Newton-Raphson method as it is called in the multivariate context, is a calculus based numerical optimization and root-finding technology. Historically, it dates back to the dawn of calculus and the time of Sir Isaac Newton (circa 1400). As described in Press, et al (1989), it is an extremely powerful optimization approach, derived from a 2nd order Taylor Series expansion. Underlying this method is a fundamental and deceptively simple mathematical insight-- any arbitrary nonlinear function can be locally approximated by a quadratic expression. Variants of the original method form a large family of nonlinear optimization techniques (Press, et al 1989, Judd 1999).

The derivation of the Newton-Raphson method is short and relatively straightforward. Let x be a vector of decision variables and f(x) be an arbitrary multivariate function of these variables. A 2nd order Taylor Series expansion (ignoring the remainder term) of f(x) around x. is then given by equation (51).

$$(51) \qquad f(x_{n+1}) \cong f(x_n) + f'(x_n)(x_{n+1} - x_n) + \frac{f''(x_n)(x_{n+1} - x_n)^2}{2}$$

Here, we introduce the subscript "n" to denote the iteration number. In vector notation, this expression can be rewritten as (52)

$$(52) \qquad f(x_{n+1}) \cong f(x_n) + (x_{n+1} - x_n)^T F_n + \frac{1}{2}(x_{n+1} - x_n)^T H_n (x_{n+1} - x_n)$$

where: $F_n = \nabla f_x = \frac{\partial f(x)}{\partial x}$ is the gradient, or vector of first partial derivatives, for the function evaluated at $x=x_n$ and $H_n = \frac{\partial^2 f}{\partial x \partial x}$ is the Hessian matrix, or matrix of second partial derivatives, also evaluated at $x=x_n$.

Maximizing the left-hand side of equation (52) with respect to x_{n+1} yields the following vector expression of first order conditions:

$$(53) \qquad F_n + H_n \times (x_{n+1} - x_n) = 0$$

Using matrix algebra and solving this equation for x_{n+1} gives us the well-known Newton-Raphson algorithm (54)

$$(54) \qquad x_{n+1} = x_n - H_n^{-1} F_n$$

The expression for the Newton-Raphson iterative solution approach is detailed in equation (54). In the Newton-Raphson algorithm, the vector of estimated choice variables (x) in the next iteration (n+1) is obtained by post-multiplying the inverse of the Hessian matrix by the gradient vector and then subtracting the result from the estimated vector available in the current iteration (n). This computation is dependent on the properties of the Hessian matrix (H) and its inverse. Provided that H^{-1} exists and is negative define, this expression can be solved iteratively to obtain estimates of the decision vector (x).

Computation of the H-inverse matrix can be a resource intensive and computationally expensive process, particularly for moderate to large problems and when the H matrix is sparse or poorly scaled. As Judd (1999), Press, et al (1989) and others have advised, well-designed implementations of this algorithm do not invert the H matrix directly. Instead, clever applications of matrix algebra are used to solve for the x vector, without (directly) undertaking the matrix inversion step.

Although an exceptionally powerful technique, the Newton-Raphson algorithm can fail, and sometimes does so,"… in a spectacular fashion" (Press et al 1989). This approach is dependent on the researcher's ability to provide a starting value for the vector of decision variables, x, (a.k.a., a guess) which is reasonably close to the true solution. If the supplied starting value for the vector x, is too far from the true solution, the algorithm will fail, particularly in cases where the function being maximized is complex or ill-behaved. As with other approaches, the Newton-Raphson method is subject to a variety of numerical computation issues including scaling, truncation, round-off error and specification error. These maladies often become apparent to the researcher when the H matrix becomes singular or ill-conditioned.

Numerical and computation problems involving the H matrix are sufficiently common that a stream of research on alternative but closely relatedtimization algorithms has arisen. These research efforts have spawned an impressive number of alternative approaches, typically classified as "quasi-Newton" or "variable metric" methods (see Press, et al 1989).

For our example problem, the vector of choice variables (x) is defined as:

(55)
$$x_n = \begin{bmatrix} q_1 \\ q_2 \\ \lambda \\ \gamma_1 \\ \gamma_2 \\ S_\lambda \\ S_{\gamma 1} \\ S_{\gamma 2} \end{bmatrix}$$

The gradient vector (vector of first partial derivatives), obtained by differentiating the Lagrangian expression with respect to each of the choice variables is given by (56).

(56)
$$F = \frac{\partial L}{\partial x_i} = \begin{bmatrix} -R_1 AW0 + 2R_1 AW1 q_1 - \lambda - \gamma_1 \\ -R_2 AW0 + 2R_2 AW1 q_2 - \lambda - \gamma_2 \\ (Q - S_\lambda^2 - \sum_1^T q_t) \\ (q_{\max} - q_1 + S_{\gamma,1}^2) \\ (q_{\max} - q_2 + S_{\gamma,2}^2) \\ -2\lambda S_\lambda \\ 2\gamma_1 S_{\gamma 1} \\ 2\gamma_2 S_{\gamma 2} \end{bmatrix}$$

The bordered Hessian is the matrix of second partial derivatives. For our example problem the Hessian matrix is derived as shown in equation (57).

(57)
$$H = \frac{\partial^2 L}{\partial x_i \partial x_j} = \begin{bmatrix} 2R_1 AW1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 2R_2 AW1 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & -2S_\lambda & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 2S_{\gamma 1} & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2S_{\gamma 2} \\ 0 & 0 & -2S_\lambda & 0 & 0 & -2\lambda & 0 & 0 \\ 0 & 0 & 0 & 2S_{\gamma 1} & 0 & 0 & 2\gamma_1 & 0 \\ 0 & 0 & 0 & 0 & 2S_{\gamma 2} & 0 & 0 & 2\gamma_2 \end{bmatrix}$$

The Newton-Raphson approach iterative procedure starts with an initial vector of the decision variables supplied by the research. The iterative process is continued until a pre-set convergence or stopping criteria is achieved. At an optimal point, the gradient vector should be equal to zero. Judd (1999) recommends a combined stopping criteria which considers the both the norm of the gradient vector and the change in the solution vector from one iteration to the next.

Appendix 7. Lambda Search Algorithm

Appendix 5 describes the analytical solution of the optimal dynamic dispatch problem. This appendix illustrates a simplified version of the dynamic dispatch problem and illustrates its solution using a calculus based traditional approach, known as lambda search.

As described previously, the hydropower plant operator is faced with a challenging dynamic optimization problem. Given the amount of water available for release and the anticipated price of electricity over a particular time horizon (T), the plant operator must decide how much water to release for generation in each period (t) in order to maximize revenue. Typically, the total amount of water available for release (Q) over the planning horizon is fixed and known. The vector of prices (R) over the planning horizon (T) is assumed or anticipated, based on prior experience and knowledge.

Ignoring (for now) the other constraints shown in the more general optimal dynamic dispatch problem, we can write a streamlined version of the problem as equations (58) through (59). For purposes of this exposition we will assume the operator will release all of the water planned for release during the planning horizon. We can then eliminate the inequality sign from equation (59).

(58) $$\textit{Maximize} \quad \sum_{1}^{T} R_t \, p_t(q_t)$$

subject to:

(59) $$\sum_{1}^{T} q_t = Q$$

Where: R_t = price ($/MWh) at time (t)
p_t = generation (MW) at time (t)
q_t = release (cfs or af) at time (t)
Q = total release (af).

The fully specified relationship between generation, release and head (see Appendix 3) is described by equation (60).

(60) $$p_t = \frac{\gamma \times eff \times [(elev_t - W0)q_t + W1q_t^2)]}{fptokw \times 1000}$$

where: $\gamma = 62.40$, specific weight of water at 50 degrees Fahrenheit (lbs/ft^3).
eff_i= efficiency factor[3] (dimensionless) for the generator.
elev = reservoir elevation (ft above mean sea level).
fptokw = 737.5, foot-pounds to kilowatt conversion factor (kW/(ft-lbs/sec)
w0 = 1708.186, height of the tailwater (ft above mean sea level) when Q=0.0
w1 = 0.0070, rate of change in tailwater elevation as release changes (ft/cfs)

Incorporating this information, we can then write out a specific relationship for the simplified dynamic economic dispatch problem as (61).

(61) $$Maximize \quad \sum_{1}^{T} R_t \frac{\gamma \times eff \times [(elev_t - W0)q_t + W1q_t^2)]}{fptokw \times 1000}$$

subject to:

(62) $$\sum_{1}^{T} q_t = Q$$

This type of optimization problem can be addressed using the method of Lagrange. The Lagrangian expression for this constrained maximization problem is written as (63).

(63) $$L = \sum_{1}^{T} R_t \frac{\gamma \times eff \times [(elev_t - W0)q_t + W1q_t^2)]}{fptokw \times 1000} + \lambda(Q - \sum_{1}^{T} q_t)$$

We can solve this problem by finding the q_t's and the λ which maximize the Lagrangian function. Before proceeding, we can further simplify this expression. Exploiting the many constants in (63), we will form a simpler constant as shown in (64).

(64) $$Let \quad A = \frac{\gamma \times eff}{fptokw \times 1000}$$

If we make this substitution, the Lagrangian expression becomes the somewhat less complicated expression shown in (65).

(65) $$L = \sum_{1}^{T} R_t \times A \times [(elev_t - W0)q_t + W1q_t^2)] + \lambda(Q - \sum_{1}^{T} q_t)$$

[3] In this application the efficiency (eff) is represented as a constant. More generally, efficiency may vary as a function of release and head.

The first order necessary conditions (FOCs) for a maximum are that each of the partial derivatives of the Lagrangian function must equal zero. The first order conditions for this expression are shown in equations (66) through (71).

(66) $$\frac{\partial L}{\partial q_1} = R_1 A[elev_1 - W0) + 2W1q_1] - \lambda = 0$$

(67) $$\frac{\partial L}{\partial q_2} = R_2 A[elev_2 - W0) + 2W1q_2] - \lambda = 0$$

(68) $$\frac{\partial L}{\partial q_3} = R_3 A[elev_3 - W0) + 2W1q_3] - \lambda = 0$$

(69) $$\vdots$$

(70) $$\frac{\partial L}{\partial q_T} = R_T A[elev_T - W0) + 2W1q_T] - \lambda = 0$$

(71) $$\frac{\partial L}{\partial \lambda} = Q - \sum_1^T q_t = 0$$

Analytic solution of this system of FOCs would be difficult if not impossible, even for low-dimension (low T) problems. The form of this problem and the resulting set of FOCs can be exploited and solved using a relatively straightforward numerical technique known as lambda search (Wood and Wollenberg 1996).

By examining the set of FOCs we can identify three useful features of these expressions. First, the single variable lambda (λ) is common to the first 1..T expressions. Second, the FOCs are of a relatively tractable form which can readily be solved for q_t if λ is known. Finally, all of the q_t's must sum up to exactly equal Q, at an optimal point. These three characteristics allow us to apply the lambda search numerical algorithm.

The lambda search algorithm and its application to dispatch problems is rather well established and is described in Wood and Wollenberg (1996). The use of this algorithm and much of the discussion which follows draws heavily from that source.

The lambda search algorithm, summarized in Figure 21, begins with a starting value for λ which is typically developed by an informed guess. Given this starting λ value, the expressions for the (T) first derivatives can be used to compute a set of output levels for each period (t). In order to meet the water use constraint, all of the q_t values should sum exactly to the amount of water available

for release (Q). In practice, there is usually some error which we will define as $\varepsilon=Q-\Sigma q_t$. We would be satisfied if the absolute value of this error ($|\varepsilon|$} was less than some arbitrarily small value which we will call the convergence tolerance (ctol). In the default case, a tolerance level of 0.01 is employed. It is unlikely that $|\varepsilon|<$ctol on the first iteration unless we make an extremely lucky guess for the value of λ! Since the probability of this happening is virtually zero, the algorithm requires a minimum of at least 2 iterations. On the second iteration a simple heuristic or rule is used to "set" the value of λ. This heuristic works as follows; if $\varepsilon>0$ set $\lambda_2=\lambda_1*0.90$, if $\varepsilon<0$ set $\lambda_2=\lambda_1*1.10$,

At the end of the second iteration, if $|\varepsilon|<$ctol, the algorithm terminates and the result is written to an output window. In the event that $|\varepsilon|>$ctol, a new value of λ is projected.

If more than 2 iterations are required, a more sophisticated approach to projecting λ is employed. Because there are constraints on the values that q_t can take, there are discontinuities in λ. Due to these discontinuities, interval bisection (Press et al 1989) is used to identify a new value of lambda for iterations 3 and higher. This new or "projected" value of λ is used in the subsequent iteration.

These iterations or loops continue until the difference between the sum of the q_t 's and Q is driven to within the user specified convergence tolerance. For logical reasons, the q_t's are constrained to remain nonnegative during these iterations.

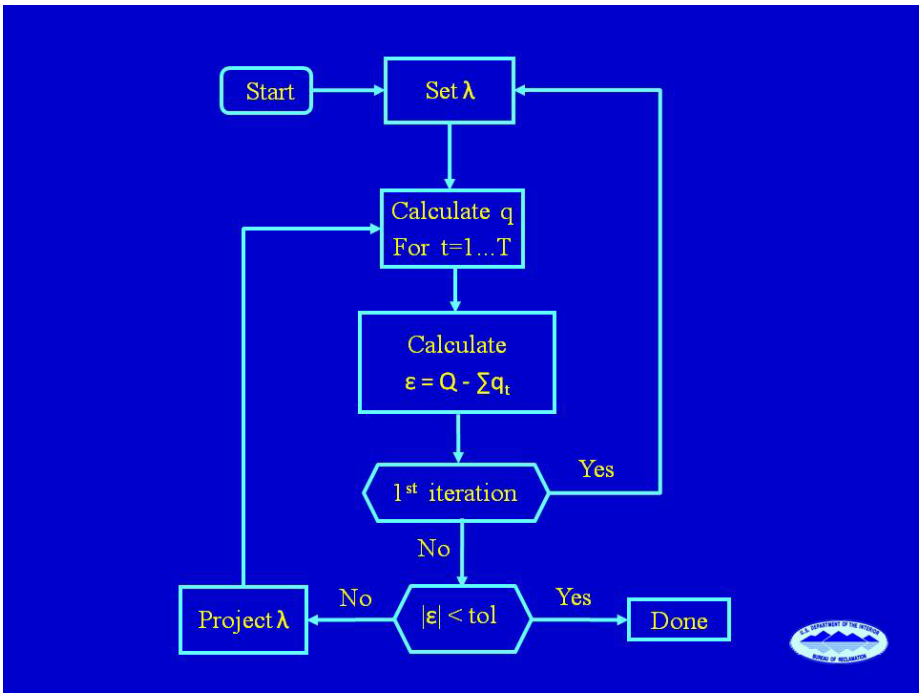


Figure 21. The Lambda search algorithm.

Potentially, the iterative process used in this algorithm can fail due to oscillations, round-off error, truncation or incorrect specification of the problem. In this application, the number of iterations is limited to some pre-set maximum to prevent the process from continuing forever should such a failure occur.

Although failures are certainly possible, our experience shows this algorithm converges very rapidly for this particular type of optimization problem.

Appendix 8. Real Coded Genetic Algorithm

Introduction

Genetic algorithms (GAs) are almost certainly the first evolutionary algorithms ever described in the literature. The GA is an optimization approach which is based on genetic principles and natural selection. A GA guides a population composed of many individuals as it evolves under specified selection rules towards a state of maximum fitness.

The method was originally developed by John Holland (1975) starting in the 1960's. Work by De Jong (1975), one of Holland's graduate students, pioneered the practical application of GA's for optimization which stimulated many threads of related research. An applied study by another one of his graduate students, David Goldberg (1989), spurred further interest in GA's. Since then, GA's have since been applied in nearly every discipline making use of optimization techniques, with Rajkumar, Vekara and Alander (2008) citing a staggering 20,488 applications. An amazing number of books have been written on GA's as typified by Michalewicz (1996), Michalewicz (2010), Haupt and Haupt (2004) and others. Pertinent to this research, Rajkumar, Vekara and Alander (2008) report 948 published GA applications in the realm of power systems engineering.

Binary GA

The original GA's described by Holland (1975) and others, utilizes a binary encoding scheme to represent numerical values and is often referred to as binary GA. These original GA's code the values of the choice variables into strings of 1's and 0's which mimics the manner in which genes are stored on the chromosome. All subsequent manipulations, such as crossover and mutation, are then carried out on these binary encoded strings. The resultant new chromosomes are then decoded from their binary form to return the actual values of the choice variables.

Like other evolutionary algorithms, the GA has a number of notable advantages over traditional (calculus based) optimization methods. It can accommodate continuous, discrete, nonlinear and complex objective functions as well as many forms of constraints. GA's do not rely on gradient information and the ability to calculate a derivative is not required for implementation. GA's are more likely to identify a global extrema and less prone to converge on a local optima. This

approach is especially well suited for complex optimization problems characterized by multiple local extrema.

Real Coded GA

Real coded genetic algorithms (RCGA) are a continuous or real valued variant of the original (binary) GA's. Real coded genetic algorithms are naturally suited for optimization of real or continuously valued objective functions. RCGA algorithms were employed exclusively in this research effort because they are computationally more efficient and because they are better suited to the test problems examined.

Like binary GA's, the RCGA approach is based on a virtual population of np-independent individuals. The population lives and evolves over a number of generations. During each generation, individuals are selected from the population to become potential parents. The selected parents may successfully reproduce to produce offspring which have some probability of undergoing mutation. The resulting offspring and/or their more fit parents are recruited into the next generation, surviving to potentially reproduce in subsequent generations. Over successive generations, the population becomes increasingly fit —thereby identifying the optimum (minimum or maximum) of a function.

In terms of efficiency and speed, RCGA algorithms have a clear-cut advantage over binary GA's. RCGA do not require encoding and decoding of the choice variables. Particularly for large dimension problems, this can be a tremendous speed advantage. Representation of real-valued variables as binary strings requires the analyst to make a tradeoff between the precision and string size. As the numeric precision is increased, considerably longer binary strings and additional memory are required. High precision characterizations necessarily increase storage memory requirements along with the computational overhead associated with reading and writing these binary strings. RCGA, which characterizes real-valued decision variables as..real-valued variables, does not suffer from this problem.

RCGA Terms

There are several GA specific terms commonly used in the literature. Among these are the following.

- Fitness function- objective function value plus penalties, if any.
- Fitness- value of the fitness function
- Current fitness− an individual's (own) fitness
- Population best – best fitness achieved by any individual in the population

Individual Components

Each of the np individuals in the virtual population consists of the following components, where d is the number of dimensions in the problem:

- Coordinates of its current position: $x=(x_1 \ldots x_d)$
- Current fitness

In the context of GA's, the vector of decision variables is called a chromosome. An individual decision variable is termed a gene.

Operationally, each individual is typically coded as either an object, in object oriented programming languages such as C#, or as a record type.

Basic RCGA Algorithm

The basic RCGA algorithm is relatively straightforward as illustrated in Figure 22. First, each of the np members of the population is created, their positions initialized in the search space and their fitness evaluated and stored. The RCGA iterative evolutionary process then begins. During each iteration or generations:

a. parent chromosomes are selected for potential reproduction;
b. with some probability, the selected parents successfully reproduce via a crossover procedure (crossover is similar to sexual reproduction in that each parent donates part of its chromosomes to the offspring);
c. with a given probability, each offspring is subject to mutation;
d. the fitness of each offspring is evaluated and stored;
e. using a recruitment strategy, the resultant offspring and parents are recruited into the next generation; and,
f. the best fitness of the population is updated.

At the end of each generation, a test is applied to determine if the population has converged. If the population has converged, the iterative process is terminated and the results are reported. If the population has not converged, a new iteration is undertaken. This process continues until the either the population has converged or the maximum number of iterations has been completed.

There are a variety of available schemes for parental selection. The most frequently encountered approaches are the roulette wheel and the tournament methods. The roulette wheel approach can only be used when the fitness function does not change sign. In other words, it can only be used if the value returned by fitness function is strictly positive, or strictly negative. The fitness functions for the three experimental test functions and the power system applications can and do change signs, negating the utility of this approach. Owing to practical details such as this, the tournament selection approach was employed in this research effort, as it has been for many other applications.

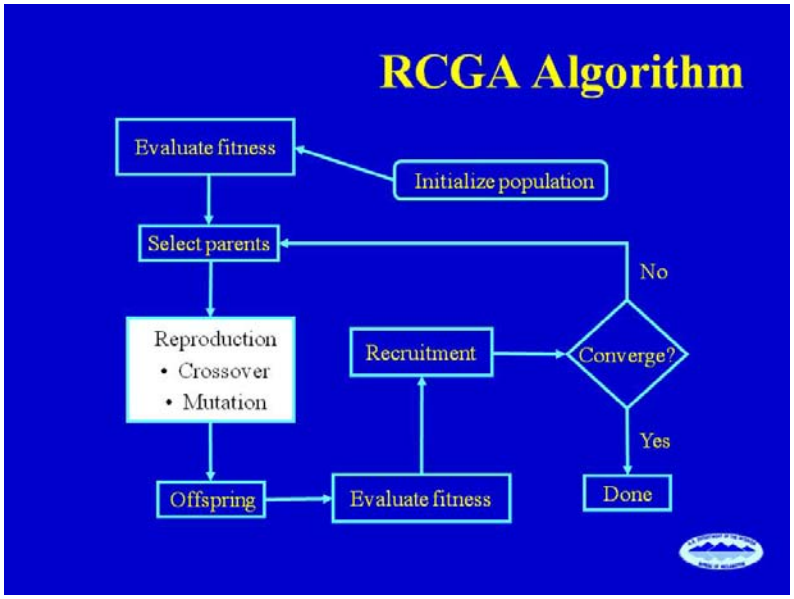


Figure 22. The basic RCGA algorithm.

Tournament selection is readily coded and executes very quickly. In the tournament selection approach, two individuals are chosen at random from the population. The chosen two individuals compete and the one with greater fitness is retained as a potential parent. Two more individuals are then randomly selected from the population and the fitter of the two is retained as a potential parent. The two potential parents which result then enter the reproduction process.

There are many different reproductive strategies described in the GA literature. Reproduction is similar to sexual reproduction in mammals and is characterized by the potential successful exchange of genetic material with a relatively high probability. If the parents successfully exchange their genes, their offspring have some the potential for mutation, which may be fitness improving or fitness degrading, with a much lower probability.

All of the crossover and mutation approaches described subsequently are designed for two parents producing one or two offspring. As in the natural world, some reproductive scenarios involve two parents producing a single offspring, two parents producing two offspring, two parents producing multiple offspring and even more than two parents producing a single offspring. Each of these schemes has its proponents and attendant niche literature. Other than practicality, there seems to be little evidence for selecting a particular scheme. For purposes of this research effort, the reproduction schemes considered were limited to two parents producing two offspring, or two parents producing a single offspring.

Parental traits are conveyed to the potential offspring via the so called crossover procedure. In the crossover process both parents intermingle their genes to produce offspring. Numerous approaches have been developed to simulate this process. In the context of RCGA, the arithmetic crossover approach

(Michalewicz 1996), the Laplace crossover approach (Deep and Thakur 2007), the linear crossover approach (Wright 1991) and the heuristic crossover approach (Michalewicz 1996) are routinely encountered in the literature. These four crossover approaches were implemented for this research effort. For reasons of brevity, only the Laplace approach is described.

The Laplace crossover approach was developed by Deep and Thakur (2007) and exploits the properties of the Laplace statistical distribution. Initially, the two prospective parents are subjected to a probability of reproduction test. The probability of (successful) reproduction is typically set at a relatively high level, with many researchers using a value of 0.90 for this parameter. A uniformly distributed random number between 0 and 1 is then generated. If this random value is less than or equal to the probability of reproduction, the two parents may reproduce. If it is not, the two parents fail to produce any offspring. Depending on their fitness ranking and the recruitment approach employed, their lineage may not continue into future generations. Assuming they are allowed to reproduce, another uniformly distributed (0,1] random number (ux) is generated. This value is used to generate a random value β which follows the Laplace distribution as shown in equation (72).

$$\beta = \mu - b * sign(ux - 0.5) * \ln(1 - 2 * |ux - 0.5|) \tag{72}$$

In this equation b which is restricted to (b>0) is the scale (dispersion) parameter for the Laplace distribution and μ is the mean or location parameter. Larger b values produce a random variable β, having a greater dispersion around the parent genes. Smaller b values produce a random variable β, which has a smaller dispersion around the parent genes.

Two offspring are then generated by the following expressions.

$$\begin{aligned} y_i^{(1)} &= x_i^{(1)} + \beta \left| x_i^{(1)} - x_i^{(2)} \right| \\ y_i^{(2)} &= x_i^{(2)} + \beta \left| x_i^{(2)} - x_i^{(1)} \right| \end{aligned} \tag{73}$$

Where: i = dimension
$x^{(J)}$ = parent J {1, 2}
$y^{(J)}$ = offspring J {1, 2}

Assuming the parents successfully reproduce, their offspring are subject to a small probability of mutation. Mutation serves to increase the diversity of the solutions. Mutation is nothing more than a random change of selected genes on the chromosome. The effects of a mutation may be fitness enhancing or fitness degrading.

There are a large number of mutation rules and described in the literature, some of which are quite ingenious. A sampling of mutation approaches include; Gaussian

mutation, nonuniform mutation (Michalewicz 1996), power mutation (Deep and Thakur 2007), uniform mutation (Michalewicz 1996) and boundary mutation (Michalewicz 1996). In the context of RCGA, uniform mutation and nonuniform mutation (Michalewicz 1996) are the two approaches predominately employed. Both of these mutation approaches were implemented for this research effort. For the sake of brevity, only the latter mutation approach will be described.

The genetic material to be mutated at the k^{th} generation is denoted as x, x_i is bounded by $\{l_i, u_i\}$ where l_i is the lower bound and u_i the upper bound on x in dimension (i). The mutated value of x is given by equation (74).

(74) $$x_i^M = \begin{cases} x_i + \Delta(k, u_i - x_i) & if\ \tau = 0 \\ x_i + \Delta(k, x_i - l_i) & if\ \tau = 1 \end{cases}$$

In equation (74), τ is a random binary digit that take on the value of either 1 or 0. The value of the function Δ is determined by equation (75).

(75) $$\Delta(k, y) = y\left(1 - \alpha^{\left(1 - \frac{k}{T}\right)^b}\right)$$

Where: α = a uniform random number [0,1]
T= maximum number of generations
b = parameter determining the degree of non-uniformity
k = generation number

The equation for $\Delta(k,y)$ returns a value in the range [0,y] so that the probability of returning a number close to zero increases as k, the number of generations, increases. During the initial generations (low value of k), the non-uniform mutation approach promotes a uniform search and in later generations (high values of k) the search space contracts, leading to a more localized search (Michalewicz 1996).

Recruitment, sometimes known as replacement, is the process of determining which individuals from the offspring population and the parent population will survive into the next generation. There are a wide variety of recruitment approaches, which have evolved over time (see Reeves 2010 p. 71 for a summary). The traditional (simple) approach, the Elite 1 approach (Bucknall 2002) and more generally, the Elite k approach.

The traditional approach to recruitment is fairly straightforward—only the offspring survive into subsequent generations. While easily implemented in code, there is a distinctive logic flaw inherent with this approach. In the traditional approach there is a probability the individual with the highest fitness will be

eliminated from the gene pool, slowing the evolutionary process and the search for an optima.

The Elite 1 approach preserves genetic material from the fittest individual in the gene pool. This method is used extensively in applied research and is fundamentally effective. In the Elite 1 recruitment approach, following reproduction, the parents are ranked from highest fitness to lowest fitness and the offspring are ranked from highest fitness to lowest. The parent individual with the highest fitness (the Elite 1) replaces the lowest ranked offspring, provided it is of superior fitness. The remaining offspring and the Elite 1 individual, survive into the next generation.

As might be anticipated, there are many potential variations on the Elite 1 approach. Limiting the population size to NP, the retained Elite fraction may vary from k=2 up to NP. Haupt and Haupt (2004, p. 62) present an example in which the top four elites (k=4) are retained in the next generation. The limiting case is the Elite NP approach, in which the fittest NP individuals from the combined parent and offspring pool, are selected for survival into the next generation. The convergence speed characteristics of these approaches improve as k increases to NP, however the diversity of the potential solutions is diminished and the likelihood of spurious convergence, or premature convergence, at a local optima, also increases.

In aggregate, the RCGA approach clearly embodies the notion of evolutionary progression or, “survival of the fittest.” The more fit individuals reproduce and pass their traits along to future generations. As in the natural world, the less fit individuals die and their inferior traits are expunged from the gene pool.

Appendix 9. Differential Evolution

Introduction

Differential evolution (DE) is one of the more recently described global heuristic optimization methods. As described on their website (www.icsi.berkeley.edu/~storn/code.html), it was jointly developed by Storn and Price (1995, 1997) and also Price and Storn (1997). The newly described DE algorithm managed to finish 3rd at the First International Contest on Evolutionary Computation (1stICEO) which was held in Nagoya, Japan in May 1996. Since that time, there have been an impressive number of DE applications encompassing at least three books (Price, Storn and Lampinen 2005, Feoktistov 2006, Chakraborty 2008 as well as several hundred published articles (see Neri and Tirronen (2010) for an overview).

Description of DE

The DE approach is based on a virtual population of np-independent individuals. During each generation, these individuals reproduce and undergo selection. Only the most-fit individuals in the population survive to reproduce in the next generation. Over successive generations, the population becomes increasingly fit —thereby identifying the optimum (minimum or maximum) of a function.

Although computationally intensive, DE has a number of notable advantages over traditional (calculus based) optimization methods. It can accommodate continuous, discrete, nonlinear and complex objective functions as well as many forms of constraints. DE does not rely on gradient information and the ability to calculate a derivative is not required for implementation. DE is more likely to identify a global extrema and less prone to converge on a local optima. This approach is especially well suited for complex optimization problems characterized by multiple local extrema.

DE Terms

There are several DE specific terms commonly used in the literature. Among these are the following.

- Fitness function- objective function value plus penalties, if any.
- Fitness- value of the fitness function
- Current fitness− an individual's (own) fitness
- Global best (g) – best fitness achieved by any individual in the population

Individual Components

Each of the np individuals in the population consists of the following components, where d is the number of dimensions in the problem:

- Coordinates of its current position: $x=(x_1 \ldots x_d)$
- Current fitness

Operationally, each individual is coded as either an object, in object oriented programming languages such as C#, or as a record type.

Basic DE Algorithm

The basic DE algorithm is amazingly simple as illustrated in Figure 23. First, each of the np particles in the population is created, their positions are initialized in the search space and their fitness evaluated. The DE iterative evolutionary process then begins. During each of these iterations or generations, (a) each of the 1…np particles reproduces, (b) each parent individual is compared to the resultant offspring and the fitter of the two survives into the next generation, and, (c) the global best fitness of the population is updated.

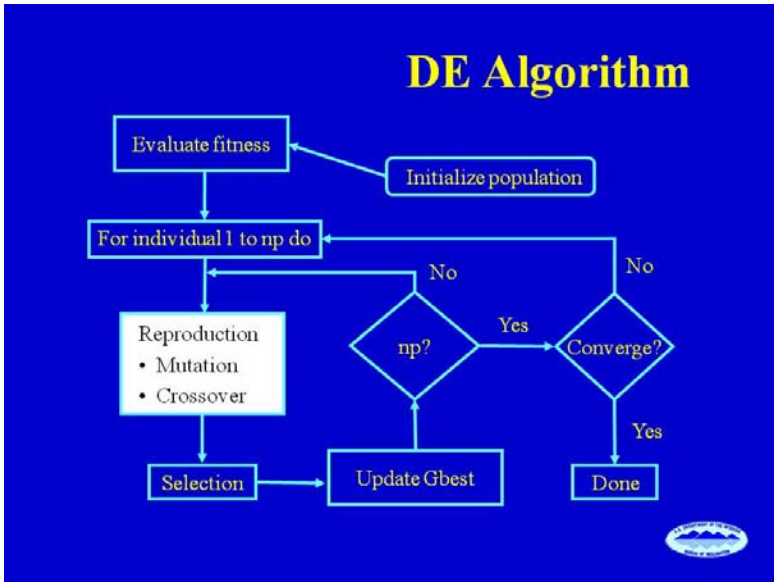


Figure 23. The basic DE algorithm.

At the end of each generation, a test is applied to determine if the population has converged. If the population has converged, the iterative process is terminated and the results are reported. If the population has not converged, a new iteration is undertaken. This process continues until the either the population has converged or the maximum number of iterations has been completed.

There are a large number of mutation rules and crossover approaches described in the literature, some of which are amazingly ingenious. A shorthand approach for describing and categorizing these variants has evolved. The notation DE/x/y/z is often used for this purpose. In this notation, x is used to specify the vector to be mutated which can be "Rand" (a randomly chosen member of the population) or "Best" (the member of the population with the current best fitness), y represents the number of difference vectors used, and, z denotes the type of crossover scheme employed. The most common crossover variant is the "Bin" or binary crossover approach.

For purpose of this Appendix, one of the most common mutation and crossover approaches, the DE/Rand/1/Bin approach, will be described. Interpretation of this shorthand notation indicates this method employs the "Rand" random method for selecting members of the population for the mutation process, uses "1" one difference vector in the mutation phase and uses the "Bin" or binary crossover method.

The DE/Rand/1/Bin variant of DE is illustrated in equations (76) and (77). Equation (76) describes the "Rand/1" mutation scheme.

(76) $$d_j[i] = p_j[r1] + F(p_j[r2] - p_j[r3])$$

Where:

d =offspring or donor individual
j=dimension index
i = individual index
r1, r2, r3 = random integer[4]
p = parent individual
F = scale parameter.

The offspring or donor vector is constructed from three randomly chosen and mutually exclusive members of the population, scaled by the parameter F. The scale parameter F is generally chosen in the range 0.1 to 1.0. Rahnamayan and Wang (2008) recommend a value of 0.50. Optimal values of F are explored by Pedersen (2010).

Parental traits are conveyed to the potential offspring via the so called crossover procedure. In DE, the most commonly encountered crossover process is the independent binomial experiment, or binary "Bin" crossover method. This crossover approach is shown in equation (77).

[4] Where $i \neq r1 \neq r2 \neq r3$.

(77) $$d_j[i] = \begin{cases} p_i, & if\ rand_i > CR \\ d_i, & if\ rand_i \le CR \end{cases}$$

where:

d = offspring or donor
p = parent individual
j=dimension index
i = individual index
$rand_i$ = uniform random deviate
CR = crossover parameter

In each dimension, a parent's traits are passed to the offspring if a uniform randomly drawn value exceeds the value of CR, the crossover parameter, a binary decision process. If the uniform random value is less than or equal to CR, the traits from the mutation process are retained by the offspring. The crossover parameter, CR is generally chosen in the range 0.10 to 1.0. Rahnamayan and Wang (2008) recommend a value of 0.90. Optimal values of CR are explored by Pedersen (2010).

In aggregate, the traits of the potential offspring are determined by mutation and crossover. Using these traits or x-values, the fitness of each offspring is evaluated and then stored.

Following reproduction, the fitness of the offspring is compared with the fitness of the parent individual in a process termed, selection. In the DE algorithm, selection follows the straightforward elite selection process as illustrated in equation (78),

(78) $$P_{i,t+1} = \begin{cases} p_i, & if\ p_i \ge d_i \\ d_i, & if\ d_i > p_i \end{cases}$$

where:

$p_{i,t+1}$ = member of next generation
d = offspring or donor
p = parent individual

As shown in equation (78), the process of selection embodies the notion of evolutionary progression or, "survival of the fittest." In DE selection, the fitness of the parent and the offspring are compared and only the most fit of the two survive, reproduce and pass their traits along to future generations. As in the natural world, the less fit individuals die and their inferior traits are expunged from the gene pool.

In comparison to other algorithms described in this document, DE has several practical advantages. First, it is simple and efficiently coded with minimal

memory requirements. Second, the selection criteria described in equation (78) ensures that each individual passing to the next generation is *at least* as fit as its parent. The evolutionary process always moves towards the optima. This is a particularly desirable feature of this algorithm.

Appendix 10. Particle Swarm Optimization

Introduction

Particle swarm optimization (PSO) is a global heuristic optimization method. Kennedy and Eberhart (1995) reportedly developed the concept by observing the behavior of flocking birds. Since that time, there have been an impressive number of PSO applications encompassing at least three books (Kennedy and Eberhardt 2001, Engelbrecht 2005, Clerc 2006) and over one thousand published articles.

Description of PSO

The PSO approach exploits the behavior of np-independent virtual particles, which "fly" through the search domain, have a memory and are able to communicate with other members of their "swarm." Each particle has a single purpose—to better its fitness—and thereby identify the optimum (minimum or maximum) of a function.

Although computationally intensive, PSO has many advantages over traditional optimization methods. It can accommodate continuous, discrete, nonlinear and complex objective functions as well as many forms of constraints. PSO is more likely to identify a global extrema and less prone to converge on a local optima. This approach is especially well suited for complex optimization problems characterized by multiple local extrema.

PSO Terms

There are several PSO specific terms commonly used in the literature. Among these are the following.

- Fitness function- objective function value plus penalties, if any.
- Fitness- value of the fitness function
- Personal best (p)− a particle's (own) best fitness
- Global (or neighborhood) best (g) – best fitness achieved by the swarm (or neighborhood sub-swarm)
- Velocity (v)− change in location from one iteration to the next along a single dimension

Individual Components

Each of the np particles in the swarm consists of the following components, where d is the number of dimensions in the problem:

- Coordinates of its position: $x=(x_1 \ldots x_d)$
- Current velocity: $v=(v_1 \ldots v_d)$
- Personal best position: $p=(p_1 \ldots p_d)$
- Global (or neighborhood) best position: $g=(g_1 \ldots g_d)$

Operationally each particle is typically coded as either an object, in object oriented programming languages such as C#, or as a record type.

Basic PSO Algorithm

The basic PSO algorithm is relatively straightforward as illustrated in Figure 24. First, each of the np particles in the swarm is created and their positions and velocities are initialized. The PSO iterative process then begins. During each of these iterations, (a) the fitness each of the 1…np particles is evaluated, (b) the personal best and global (or neighborhood) best of each particle in the swarm are updated, and, (c) a new velocity and a new particle position are computed. A test is then applied to determine if the swarm has converged. If the swarm has converged, the iterative process is terminated and the results are reported. If the swarm has not converged, a new iteration is undertaken. This process continues until the swarm has either converged or the maximum number of iterations has been completed.

The velocity, or change in the location of each particle in a given dimension, is updated according to the rule illustrated in equation (79).

(79) $$v_d(t) = w[v_d(t-1)] + c_1 rand_1[g_d - x_d] + c_2 rand_2[p_d - x_d]$$

Where:

w = inertia coefficient
c1,c2 = cognitive and social weights
rand = uniform random value
v = velocity x = current location
g = global best p = personal best
t=iteration counter or index.

The new velocity of each particle depends on the velocity in the previous iteration, an inertia coefficient (w), the cognitive weight (c1), a social weight (c2), the particle's current location in each of the d-dimensions (x_d), two random uniform deviates, the particle's own personal best position (p_d), and the global (or neighborhood) best position (g_d).

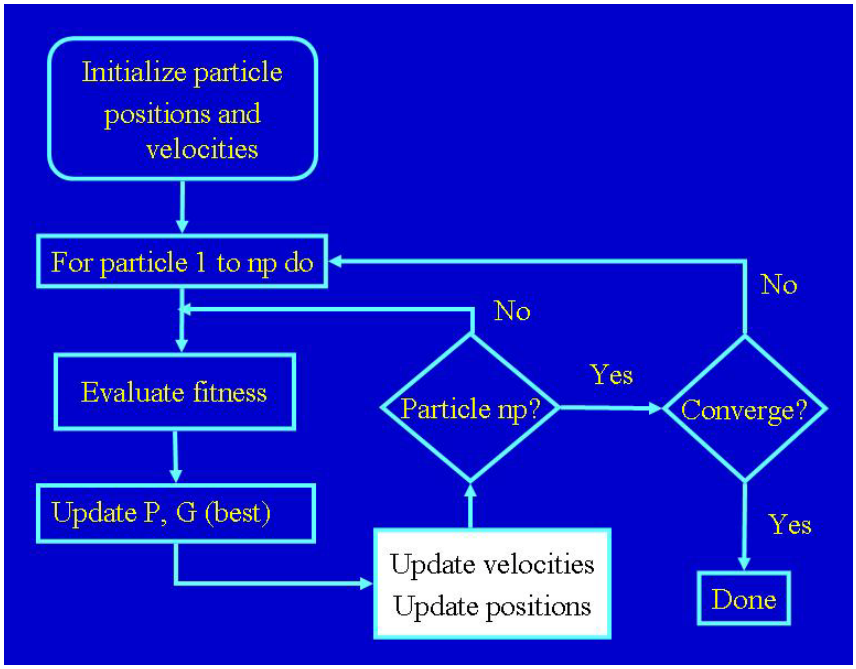


Figure 24. The basic PSO algorithm.

After the particle's velocity has been updated, its position is updated using equation (80).

(80) $$x_d(t) = x_d(t-1) + v_d(t)$$

where:

v = velocity
x = current location

As shown, each particle's new position depends on its position in the previous iteration and the new (updated) velocity.

Modified PSO

A modified version of the basic PSO algorithm was employed for the research described in this document. This modification consisted of augmenting the basic PSO algorithm with a selection mechanism, identical to that employed in the DE algorithm. The PSO selection routine ensured the individuals which survived into the next generation would be at least as fit as their parents.

The basic PSO algorithm described previously in this Appendix was fast and highly successful when applied to the unconstrained optimization problems in Phase 1 of the development process. However, when applied to the constrained dynamic economic dispatch problem in Phase 2 of the development process, it failed to achieve convergence. The basic PSO algorithm was able to quickly

locate the neighborhood of the optimal solution, but then ran for many thousands of iterations without converging to a tolerance of 1.0e-04. This convergence failure behavior was exhibited over a range of social and personal acceleration coefficient (C_1 and C_2) values and for all reasonable convergence criterion settings. Considerable effort was expended to diagnose and remedy this behavior—without appreciable progress.

Ultimately, a modified version of the basic PSO algorithm was developed for application to the constrained dynamic economic dispatch problem. The structure of the basic PSO algorithm as reported in equations (79) and (80) was retained. The population update mechanism found in the basic PSO algorithm was altered from the usual unconditional approach to a selection process. This selection process ensured that the personal best fitness of each particle was non-decreasing over each successive generation.

Following reproduction, the personal best fitness of the offspring is compared with the personal best fitness of the parent. As in the DE algorithm, selection follows the straightforward elite selection process as illustrated in equation (81),

(81) $$P_{i,t+1} = \begin{cases} p_i, & if \;\; p_i \geq d_i \\ d_i, & if \;\; d_i > p_i \end{cases}$$

where:

$p_{i,t+1}$ = member of next generation
d = offspring or donor
p = parent individual

The fitness of the parent and the offspring are compared using the approach described in Deb (2000) and only the most fit of the two survive to pass their traits along to future generations. The selection mechanism described in equation (81) ensures that each individual passing to the next generation is *at least* as fit as its parent. The evolutionary process always moves the swarm towards the optimal point. This is a particularly desirable feature and reliably leads the modified PSO algorithm to convergence.

Appendix 11. Clerc's K

Early empirical studies established the PSO algorithm could fail to converge, even when the social and personal acceleration coefficients (c_1, c_2) were properly defined. In essence, the swarm could diverge or explode, rather than converge. Two methods are in common usage to counteract this potential behavior. These are Clerc's constriction factor or coefficient (K), and, the use of an inertia weight.

For purposes of this research effort, the approach developed by Clerc and Kennedy (2002) was employed. As further described in Clerc (2006) the constriction coefficient (k) is applied as shown in equation (82) below.

(82) $$v_d(t) = k\{[v_d(t-1)] + c_1 rand_1[g_d - x_d] + c_2 rand_2[p_d - x_d]\}$$

Where:

k = Clerc's constriction coefficient
c1,c2 = cognitive and social weights
rand = uniform random value
v = velocity x = current location
g = global best p = personal best
t=iteration counter or index.

The simplest case is known as a the Type 1 coefficient which is defined as shown in equation (83)

(83) $$k = \frac{2.0}{\left|2.0 - \theta - \sqrt{\theta^2 - 4\theta}\right|} \qquad c_1 + c_2 = \theta > 4.0$$

This constriction factor improves the convergence of the PSO algorithm over time by damping particle oscillation in the neighborhood of a potential optima, while preserving the search behavior of the swarm. Its main disadvantage is that it is not as effective in promoting convergence, as for example, an inertia weight. Unlike an inertia weight however, it does not drastically diminish a swarm's ability to explore promising new solution regions, once one potential optimal point has been discovered.

Appendix 12. Random Numbers

The research effort described here is focused on algorithms and techniques which are randomly varying or stochastic in nature. These algorithms necessarily make use of random number generators (RNGs). The performance of these algorithms and the results obtained with them are critically dependent on the speed and quality of these underlying RNGs.

All of the optimization algorithms explored here make extensive use of RNGs in three important phases of their operation, (a) initialization, (b) selection and (c) search. Initialization—a finite number of particles or individuals are distributed in the d-dimensional search space. Although other methods have been proposed, the typical approach is to randomly initialize the individuals using an RNG. Selection—selection of the fittest individual, construction of a neighborhood topology or other grouping mechanisms and reproduction in the population are typically guided by a randomized choice process. These actions are all based on an RNG. Search—at each step or iteration of the algorithm, the search activity is influenced by a random component or influence. These random influences are supplied by an RNG.

Although it may prove surprising to the layperson, the descriptive term "random number generator" is largely a misnomer (see, for example, Knuth 2002 and Judd 1999). As a rule, researchers reserve the word random, "…for the output of an intrinsically random process, like the elapsed time between clicks of a Gieger counter placed next to a sample of some radioactive element" (Press et al 1989). Software based random number generators are inherently deterministic (For a given set of parameters and starting value(s), they produce identical results). For this reason, most researchers and mathematicians employ the terms quasi-random or pseudorandom to describe software based RNGs.

Many RNG implementations are based on the linear congruential generator method. A linear congruential generator is an iterative mathematical relationship of the form shown in equation (84).

(84) $$X_{t+1} = (a * X_t + c) \bmod m$$

In equation (84), *a*, *b* and *m* are integers and "mod" is the modulo operator which returns the remainder when the expression in parentheses is divided by *m*..

As shown in (84) the analyst supplies a random seed, or starting value (X_t). Given this random seed and values for *a*, *b* and *m*, a random deviate (X_{t+1}) is produced. This deviate is then used as the seed for producing the next random

value. A mathematical property of expression (84) is that it will eventually repeat itself, with a period that is no greater than *m*.

The values for *a*, *b* and *m* and the random seed must be chosen with some care. Generally, *m* is chosen to be as large an integer as possible. As a practical matter, its size is limited by the interaction of the software and the computer hardware. In ANSI C, the maximum integer value is specified to be only 32767 (corresponding to the Delphi 16 bit SMALLint type). As described further in Press et al (1989), these limitations will result in the randomly generated points lying on at most $32768^{1/3}$ or 32 planes (in three dimensional space). If *a*, *b* and *m* are not carefully selected, the points will lie on far fewer planes (Press et al 1989). As a result, the analyst could unknowingly be focusing their attention on a relatively small and discrete portion of the search region.

Modern programming languages such as Visual BASIC, FORTRAN, C/C++, Java and Borland Delphi all contain RNG implementations as part of their supplied feature set. Unfortunately, these built-in system RNGs are of uneven quality and some have readily demonstrable flaws.

The existing numerical analysis literature contains numerous and rather blunt warnings to researchers about the failings of system RNGs. The writings of Park and Miller (1988), Klimasauskas (2002), Knuth (2002) and Judd (1999) are especially accessible examples. One of the more informative assessments of this topic is provided by Press, et al (1989). They caution researchers to be very, *very* suspicious of system supplied random number generators, which often resemble equation (84). They write that, "If all scientific papers whose results are in doubt because of bad rand()s [RNGs] were to disappear from library shelves, there would be a gap on each shelf as big as a fist (Press, et al 1989, page 214).

There are quite a large number of statistical tests available for discerning the capability of RNGs to produce random sequences. These range from well-known statistical techniques such as the Chi-Square test, the runs test, the monobit test and the continuous RNG test (Vicaria 2003), to more esoteric and complex approaches such as the Knuth spectral test (Knuth 2002). A frequently employed and well known suite of RNG tests, known as the Diehard test suite, was developed by George Marsaglia (currently available for purchase from: http://www.stat.fsu.edu/pub/diehard/). A new, improved and Open Public License (OPL) version, called the DieHarder test suite, has been developed by Robert G. Brown (freely available from: http://www.phy.duke.edu/~rgb/General/dieharder.php).

Of direct pertinence to this study, Klimasauskas (2002) has employed a battery of statistical tests to examine the system RNGs supplied with a wide range of programming languages including IBM FORTRAN, Visual BASIC, Delphi Object Pascal (parent of the Delphi programming language) and Visual C/C++ as well as some spreadsheet software tools such as Microsoft Excel. He reported

that all of the commonly available system RNGs failed the standard tests for randomness, some spectacularly!

In light of this disconcerting evidence, the selection of an RNG for use in this study proved to be a nontrivial decision. It was clearly inadvisable to use the system RNG, which was known to be flawed. Conversely, the circumstances of this research did not warrant the use of a cryptographic grade RNG such as the Microsoft Crypotography API (see Vicaria 2003 for a description). To allow for scientific replication and for purposes of comparability, it seemed essential to explicitly document the RNG that was employed and to use it consistently throughout the study. After carefully considering the available options, a well-proven, if not state-of-the-art RNG was adopted for use. All of the algorithms developed during this research effort employ a Delphi coded implementation of the Mersenne Twister RNG (Matsumoto and Nishimura 1998) developed by David Butler and obtained from the SourceForge Library, http://fundementals.sourceforge.net/units.html. This algorithm is also known by its Association for Computational Machinery (ACM) identification number as algorithm MT19937.

Appendix 13. Low Discrepancy Sequences

Low discrepancy sequences are those whose points are approximately proportionally distributed within the space enclosed by a set with arbitrary boundaries. Low-discrepancy sequences are also called quasi-random or sub-random sequences. This is a possible source of confusion, since they are often used as a substitute for randomly generated sequences. Some commonly encountered examples of low discrepancy sequences are the Sobol sequence, the Neiderreiter sequence, the Weyl sequence, the Haber sequence, the Halton sequence and the Hammersley sequence.

Appendix 12 described random number generators (RNGs), their properties and some of their weaknesses. As noted there, the optimization algorithms explored in this document make extensive use of RNGs for, (1) initialization, (2) selection, and (3) search.

As research on these heuristic optimization algorithms has progressed, several authors have suggested the usage of RNGs may not be the preferred approach (Clerc 2008). Conceptually, what is desirable is not randomness *per se* but an exhaustive and systematic distribution of np-points in the d-dimensional search space. For optimization algorithm applications, while employing RNGs is convenient, RNGs have some well-known drawbacks. Clerc (2008) summarizes this problem rather well in the title to the first paragraph of his paper as, "Uniform Random Distributions: Easy but Bad."

A subset of recent research efforts has focused attention on the potential advantages of employing low discrepancy sequences in heuristic algorithms instead of the more traditional RNGs. The hypothesis is, these sequences may be better suited to usage for initialization, selection and search applications. Applied research by Richards and Ventura (2004), Pant, Thangaraj and Abraham (2009), and Uy, Hoai, McKay and Tuan (2007) certainly seems to provide empirical evidence supporting this view.

This research thread is based on the mathematical properties of low discrepancy sequences which allow them to more exhaustively and systematically span the d-dimensional search space. While these properties can be statistically demonstrated, the visual approach provides much the same intuition. Figure 25 shows plots of the first 250 points in 2-dimensions over the range (0,1) generated using the Mersenne Twister RNG (described in Appendix 12), the Neiderreiter sequence, the Weyl sequence and the Haber sequence.

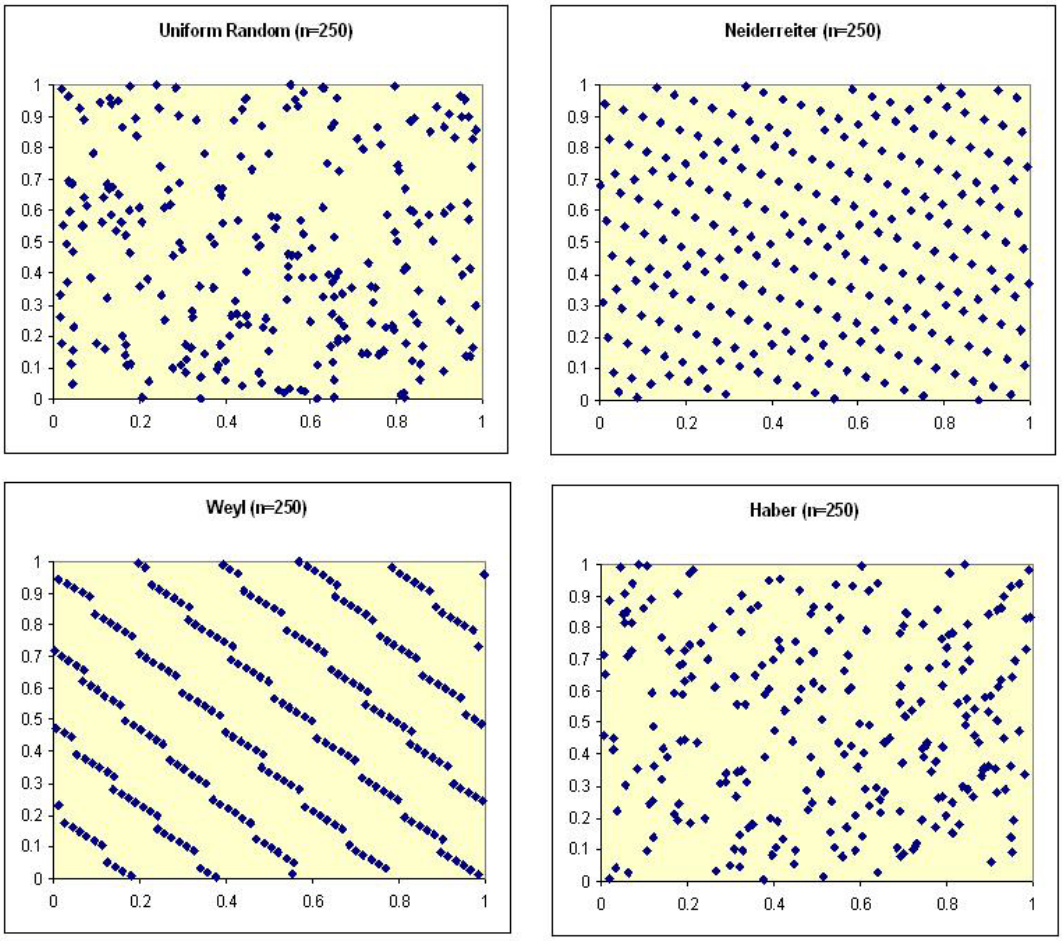


Figure 25. Plots of the first 250 points generated by four RNG methods.

A visual comparison of these plots illustrates that points generated by the low discrepancy sequences are more regularly distributed in the 2-dimentional space than those from the RNG. Compared to the RNG plot, there are fewer "gaps" between the low discrepancy points and the space between points is more even. This figure demonstrates one potential advantage of low discrepancy sequences. Other research indicates the points generated by RNGs tend to collapse around the origin as the number of dimensions increase. Certain low discrepancy sequences, such as the Sobol, have not shown this behavior in empirical studies. Both of these outcomes point to the potential advantages of employing low discrepancy sequence methods for heuristic optimization, especially in high dimensional cases.

One component of this research effort was to investigate the potentials advantages of employing some of these low discrepancy sequences. As part of that effort, computer codes for the Neiderreiter, Weyl, Haber, Halton and Torus sequences were developed. This code was based on MatLab code from the EconToolbox which accompanies Miranda and Fackler (2006). The prime numbers utilized in coding these sequences were drawn from Caldwell (2009).

Appendix 14. Test Functions for Algorithm Development

Introduction

This Appendix describes three optimization test functions which were employed in the early phases of algorithm development, including coding, testing, performance visualization and validation. Each of these functions is a previously studied 3-dimensional (3-D) continuous, unconstrained, optimization problem. The test functions selected were restricted to three dimensions to facilitate implementation and to allow for real-time visualization of the algorithm's behavior in the search space. These test functions facilitated development of the evolutionary algorithms, prior to their application to the more difficult and higher dimension electric power-related problems, which were the focus of this research.

By design, this research effort utilizes only a small and rather rudimentary subset of the universe of test functions investigated by other authors. As popularized by De Jong (1975), many existing studies examine the performance of evolutionary algorithms on a suite of optimization test functions (for example, see Mishra 2007 or Mezura-Montes and Flores-Mendoza 2009). There are numerous optimization test functions available for this purpose, some of which are exceedingly complex. A sample of the test functions encountered in this literature is described in De Jong (1975), Haupt and Haupt (2004), Engelbrecht (2005), Price, Storn and Lampinen (2005), Feoktistov (2006) and other sources.

Test Function 1—Sphere

The sphere function is one of the most rudimentary 3-D optimization problems. It is a symmetric, continuous real-valued function possessing a single (global) optimal point. This function is defined over the set of all real numbers, however a bounded search range is used in this application.

In 3-D, the equation describing the sphere function is (85).

(85) $$Z = -x^2 - y^2$$

The gradient of this test function is especially useful as a device for ascertaining the quality of a solution on convergence. For this test function, the expression for the gradient, or vector of first partial derivatives, is shown in equation (86).

(86) $$\nabla Z_x = \frac{\partial Z}{\partial x_i} = -2x_i$$

The domain for the sphere function is the real number line ($-\infty \leq x \leq +\infty$). For test purposes however, the search domain for the independent variables (x, y) was restricted to the bounded interval ($-2 \leq x \leq +2$).

The maximum value of Z for this test optimization function is Z=0.0. This maximum Z value is obtained when x=0.0 and y=0.0.

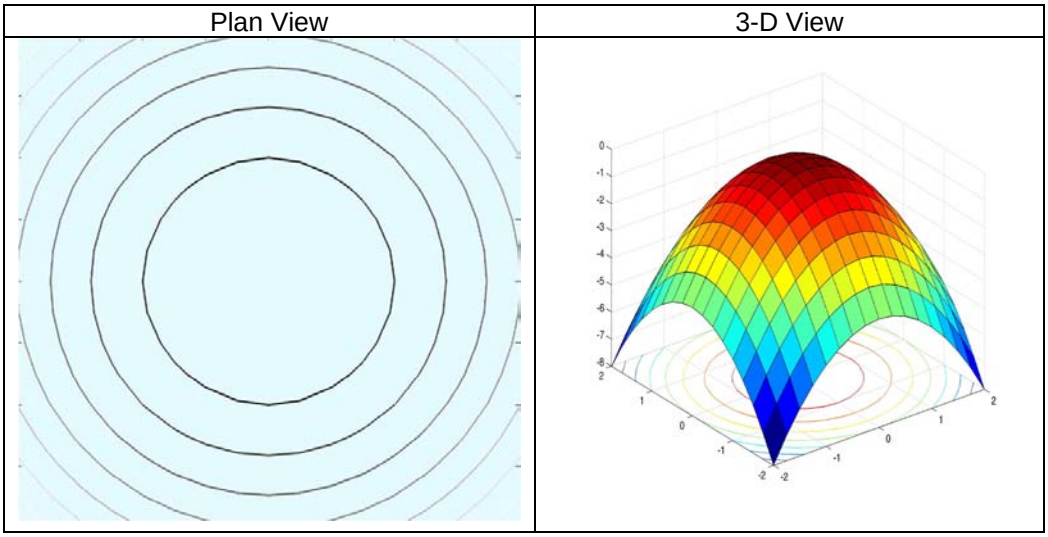


Figure 26. Plan and 3-D views of the Sphere function.

Figure 26 illustrates the plan (top) view and the 3-D view of test function 1, the sphere function. As shown in this figure, the contours are symmetric about the optimal point. The global maximum is the sole optima in the bounded search space.

The sphere function is perhaps the most rudimentary of all optimization test problems. It is symmetric about the origin, easily implemented in code and readily solved. This function was used primarily for early-stage development of the evolutionary algorithms employed in this research. This test function allowed for visual verification of algorithm functioning and effectiveness during the coding process.

Test Function 2—Ridge

The ridge function is a somewhat more complex 3-D optimization problem. It is a continuous but not a symmetric function. It has a single (global) optimal point located at the top of a ridge, bounded on either side by steep canyons. This

function is defined over the set of all real numbers, however a bounded search range is used in this application.

In 3-D, the equation describing the ridge function is (87).

(87) $$Z = x + 2ey - e^{x} - e^{2y}$$

The gradient of this test function is especially useful as a device for ascertaining the quality of a solution on convergence. For this test function, the expressions for the gradient, or vector of first partial derivatives, is shown in equations (88) and (89)

(88) $$\nabla Z_x = \frac{\partial Z}{\partial x} = 1.00 - e^{x}$$

(89) $$\nabla Z_y = \frac{\partial Z}{\partial y} = 2e - 2e^{2y}$$

The domain for the ridge function is the real number line ($-\infty \leq x \leq +\infty$). For test purposes however, the search domain for the independent variables (x, y) was restricted to the bounded interval ($-2 \leq x \leq +2$).

The maximum value of Z for this test optimization function is Z=-1.00. This maximum Z value is obtained when x=0.0 and y=0.50.

Figure 27 illustrates the plan (top) view and the 3-D view of test function 2, the so called ridge function. As shown in this figure, the contours are asymmetric about the optimal point. The global maximum is the sole optima in the bounded search space. It lies at the top of a long gently sloping ridge with canyons on either side.

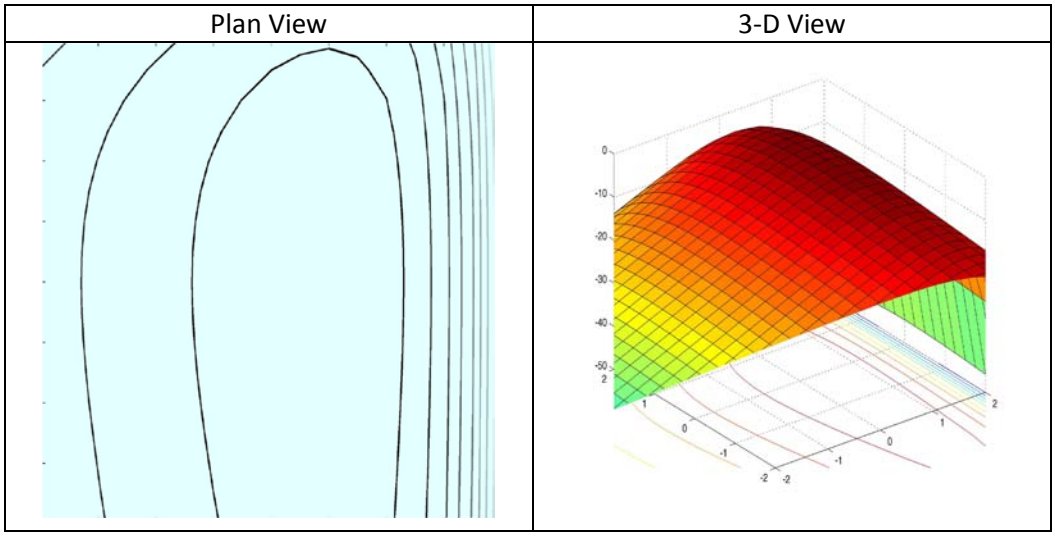


Figure 27. Plan and 3-D views of the Ridge function.

The slope of the ridge changes very gradually, often causing premature convergence for certain types of gradient based algorithms and poorly parameterized evolutionary algorithms. A single badly calculated step can send the solution down one of the precipitous canyons on either side of the ridge, causing the algorithm to fail.

In the general scheme of things, the ridge function is a relatively straightforward optimization test problem. It is easily implemented in code, although not so readily solved. This function was used as a test bed during the development of the evolutionary algorithms described in this research. This test function allowed for visual verification of algorithm functioning and effectiveness during the coding process.

Test Function 3—Alpine

The Alpine test function, as described by Clerc (2006) and Haupt and Haupt (2004), is a complex 3-D optimization problem. It is a continuous, but not symmetric function. It has a multiple local optima and a single (global) optimal point in the search space. This function is defined over the set of all real numbers and has multiple local optima in that range (as might be expected). A finite bounded search range is used in this application.

In 3-D, the equation describing the Alpine function is (90).

(90) $$Z = \sin(x) \times \sin(y) * \sqrt{xy}$$

The gradient of this test function is especially useful as a device for ascertaining the quality of a solution at convergence. For this test function, the expressions for the gradient, or vector of first partial derivatives, is shown in equation (91).

(91) $$\nabla Z_x = \frac{\partial Z}{\partial x} = \tanh(x_i) + 2x_i$$

The domain for the ridge function is the real number line ($-\infty \leq x \leq +\infty$). For test purposes however, the search domain for the independent variables (x, y) was restricted to the bounded interval ($-10 \leq x \leq +10$).

The maximum value of Z for this test optimization function is Z=7.885600724. The maximum Z value is obtained when x=7.917052686 and y=7.917052686. This point is located in the upper right-hand quadrant of the plot.

Figure 28 illustrates the plan (top) view and the 3-D view of test function 3, the Alpine function. As shown in this figure, the contours are quite complex. There are multiple local optima in the search space. The global maximum (in this bounded search space) is located at the top of Mount Blanc (as termed by Clerc 2006) or Longs Peak (as termed by Haupt and Haupt 2004), in the upper right-hand quadrant of the plot. Mount Blanc is surrounded by lesser peaks.

Encountering any of these lesser peaks will cause a gradient based algorithm to converge and announce it has located a solution. Identification of the global optima in this search space is extremely difficult.

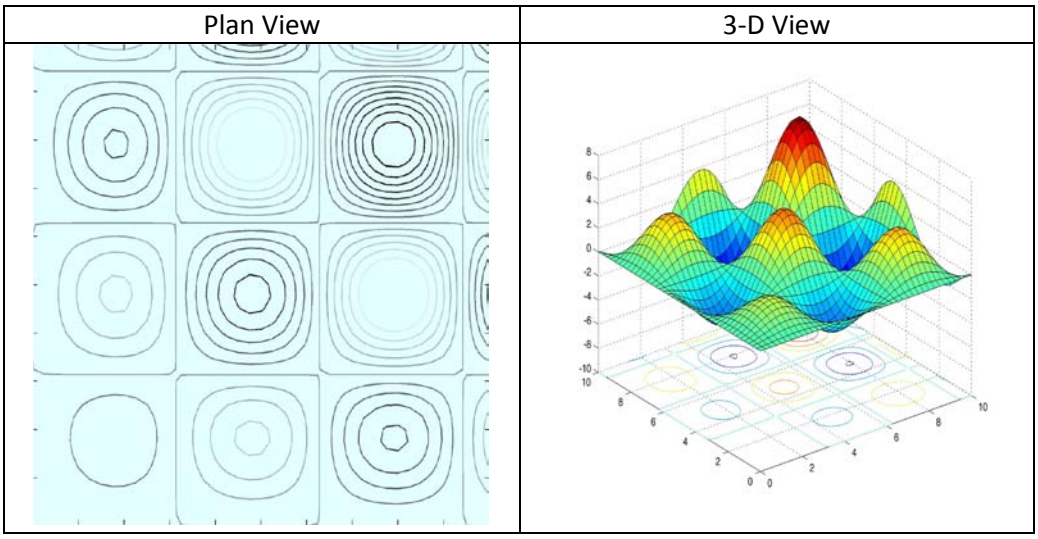


Figure 28. Plan and 3-D views of the Alpine function

In the general scheme of things, the Alpine function is a relatively complicated optimization test problem. It is quite easily implemented in code, although not so readily solved, particularly by gradient based methods. This function was used extensively during the development of the evolutionary algorithms described in this research. This test function allowed for visual verification of algorithm functioning and effectiveness in a complicated solution space.

Appendix 15. 24-Hour Price Vector

The 1-day summer and winter hourly electricity prices (avoided costs) used in this analysis are shown in this appendix. These prices span a one day period (24 hours) and were generated by the ESIM03 model (Harpman 2006).

Price units: dollars per megawatt-hour ($/MWh).

Summer		Winter	
Hour	Price ($/MWh)	Hour	Price ($/MWh)
1	45.85	1	39.49
2	45.11	2	39.07
3	44.71	3	38.99
4	44.75	4	39.28
5	45.16	5	40.16
6	46.35	6	42.21
7	48.08	7	45.12
8	49.86	8	47.41
9	53.37	9	47.60
10	56.45	10	47.23
11	58.26	11	46.77
12	59.23	12	45.85
13	59.89	13	45.02
14	60.53	14	44.24
15	61.07	15	43.80
16	61.45	16	43.98
17	61.46	17	46.19
18	60.94	18	49.26
19	59.98	19	50.15
20	59.51	20	49.41
21	59.08	21	48.40
22	57.09	22	46.38
23	51.11	23	43.46
24	48.07	24	41.37

Appendix 16. 168-Hour Winter Prices.

The hourly winter electricity prices (avoided costs) used in this analysis are shown in this appendix. These prices span a 1-week (168-hour) period, starting on Sunday and ending on Saturday, and were generated by the ESIM03 model (Harpman 2006).

Price units: dollars per megawatt-hour ($/MWh).

Hour	Price	Hour	Price	Hour	Price	Hour	Price	Hour	Price	Hour	Price	Hour	Price
1	39.42	25	39.49	49	40.10	73	40.30	97	40.19	121	39.96	145	39.72
2	38.86	26	39.07	50	39.53	74	39.76	98	39.64	122	39.39	146	39.08
3	38.73	27	38.99	51	39.43	75	39.64	99	39.46	123	39.18	147	38.88
4	38.74	28	39.28	52	39.64	76	39.83	100	39.64	124	39.28	148	38.89
5	39.13	29	40.16	53	40.54	77	40.66	101	40.42	125	39.92	149	39.30
6	40.03	30	42.21	54	42.66	78	42.66	102	42.28	126	41.56	150	40.19
7	41.38	31	45.12	55	45.78	79	45.66	103	44.88	127	43.28	151	41.35
8	42.81	32	47.41	56	48.04	80	47.84	104	47.00	128	45.03	152	42.40
9	43.65	33	47.60	57	47.89	81	47.80	105	46.96	129	45.61	153	43.26
10	43.94	34	47.23	58	47.28	82	47.22	106	46.49	130	45.67	154	43.49
11	43.85	35	46.77	59	46.57	83	46.62	107	45.88	131	45.31	155	43.25
12	43.52	36	45.85	60	45.76	84	45.83	108	44.89	132	44.36	156	42.79
13	43.11	37	45.02	61	44.88	85	45.15	109	43.88	133	43.58	157	42.24
14	42.62	38	44.24	62	44.31	86	44.64	110	43.49	134	43.21	158	41.72
15	42.30	39	43.80	63	43.82	87	44.00	111	43.20	135	42.93	159	41.39
16	42.47	40	43.98	64	43.99	88	44.31	112	43.31	136	43.04	160	41.52
17	43.78	41	46.19	65	46.35	89	46.41	113	45.01	137	43.99	161	42.61
18	46.82	42	49.26	66	49.49	90	49.22	114	47.75	138	46.41	162	44.08
19	47.60	43	50.15	67	50.49	91	50.09	115	48.38	139	46.37	163	44.75
20	47.23	44	49.41	68	49.69	92	49.17	116	47.75	140	45.21	164	44.35
21	46.42	45	48.40	69	48.70	93	48.13	117	46.93	141	44.00	165	43.68
22	44.74	46	46.38	70	46.66	94	46.18	118	45.23	142	43.27	166	43.04
23	42.62	47	43.46	71	43.58	95	43.39	119	43.01	143	42.24	167	41.91
24	40.49	48	41.37	72	41.51	96	41.43	120	41.15	144	40.84	168	40.55

Appendix 17. 168-Hour Summer Prices

The hourly summer electricity prices (avoided costs) used in this analysis are shown in this appendix. These prices span a 1-week (168 hour) period, starting on Sunday and ending on Saturday, and were generated by the ESIM03 model (Harpman 2006).

Price units: dollars per megawatt-hour ($/MWh).

Hour	Price	Hour	Price	Hour	Price	Hour	Price	Hour	Price	Hour	Price	Hour	Price
1	45.95	25	45.85	49	46.73	73	47.09	97	46.76	121	46.90	145	46.42
2	45.20	26	45.11	50	45.91	74	46.19	98	45.91	122	46.13	146	45.69
3	44.58	27	44.71	51	45.39	75	45.64	99	45.56	123	45.61	147	45.10
4	44.16	28	44.75	52	45.18	76	45.36	100	45.17	124	45.19	148	44.62
5	44.05	29	45.16	53	45.50	77	45.69	101	45.47	125	45.46	149	44.57
6	44.11	30	46.35	54	46.71	78	46.82	102	46.59	126	46.55	150	44.79
7	44.39	31	48.08	55	48.40	79	48.49	103	48.23	127	48.19	151	45.33
8	45.58	32	49.86	56	50.14	80	50.25	104	49.96	128	50.00	152	46.69
9	47.41	33	53.37	57	53.78	81	54.20	105	53.42	129	53.73	153	48.60
10	48.97	34	56.45	58	56.66	82	56.76	106	56.08	130	55.96	154	50.21
11	50.23	35	58.26	59	58.01	83	58.06	107	57.36	131	57.33	155	53.58
12	52.95	36	59.23	60	58.95	84	59.12	108	58.49	132	58.33	156	54.94
13	54.54	37	59.89	61	59.91	85	60.08	109	59.36	133	59.17	157	55.55
14	55.80	38	60.53	62	60.55	86	60.77	110	59.94	134	59.65	158	55.96
15	56.53	39	61.07	63	60.81	87	61.24	111	60.62	135	60.18	159	56.17
16	57.31	40	61.45	64	60.93	88	61.51	112	60.93	136	60.37	160	56.36
17	57.86	41	61.46	65	60.64	89	61.37	113	60.87	137	60.05	161	56.39
18	57.92	42	60.94	66	59.91	90	60.64	114	60.30	138	58.92	162	56.16
19	57.07	43	59.98	67	59.02	91	59.67	115	59.18	139	57.62	163	55.41
20	56.73	44	59.51	68	58.64	92	59.16	116	58.65	140	56.91	164	55.08
21	56.44	45	59.08	69	58.39	93	58.66	117	58.22	141	56.53	165	54.81
22	53.47	46	57.09	70	56.38	94	56.66	118	56.42	142	54.91	166	52.49
23	49.16	47	51.11	71	50.12	95	51.28	119	51.42	143	49.95	167	49.10
24	47.12	48	48.07	72	47.82	96	48.17	120	48.33	144	47.85	168	47.20

Appendix 18. Dimension and Input Experiment.

The numerical results of replicated experiments (ntrials=50) using different problem dimensions and input price vectors are reported in this appendix. Some of these results are summarized in graphic form in the main body of the report.

Table 11. Dimension and Input Results

Parameter		1 Day (24 hrs)		1 Week (168 hrs)	
		Summer	Winter	Summer	Winter
LS[1]	Mean best	127,097.33	103,653.36	921,829.41	752,472.91
	S.D. best	na	na	na	na
	Mean Iter	28	28	28	28
	Mean CPU	≤0.002	≤0.002	≤0.002	≤0.002
RCGA	Mean best	127,097.32	103,653.35	921,829.15	752,472.64
	S.D. best	1.528e-03	6.861e-01	1.831e-03	9.153e-02
	Mean Iter	635	621	5224	4904
	Mean CPU	0.198	0.195	9.860	9.258
DE	Mean best	127,097.33	103,653.36	921,829.43	752,472.90
	S.D. best	2.517e-04	1,911e-04	1.716e-04	1.767e-04
	Mean Iter	242	244	916	916
	Mean CPU	0.229	0.223	5.493	5.266
PSO	Mean best	127,097.33	103,653.36	921,829.41	752,472.88
	S.D. best	1.960e-04	1.706e-04	2.5116e-04	2.054e-04
	Mean Iter	491	492	4941	4821
	Mean CPU	0.176	0.192	11.811	11.420

[1] Lambda search is a deterministic approach and each trial produces the same outcome. The results reported here were generated by a single trial at ctol=1.0e-08.

Appendix 19. Maximum Release Constraint Experiment

The numerical results of replicated experiments (ntrials=50) using a (binding) maximum release constraint of 6,000 cfs are reported in this appendix. Some of these results are summarized in graphic form in the main body of the report.

Table 12. Maximum Release Constraint Results

Parameter		168-hour Base Case		Maximum Release Constraint 6,000 cfs (Binding)	
		Summer	Winter	Summer	Winter
LS[1]	Mean best	921,829.41	752,472.91	916,144.65	751,391.53
	S.D. best	na	na	na	na
	Mean Iter	28	28	28	28
	Mean CPU	≤0.002	≤0.002	0.016	≤0.002
RCGA	Mean best	921,829.15	752,472.64	915,545.85	751,075.56
	S.D. best	1.831e-03	9.153e-02	6.768e-01	1.030e-01
	Mean Iter	5224	4904	1534	1727
	Mean CPU	9.860	9.258	8.903	5.609
DE	Mean best	921,829.43	752,472.90	916,109.84	751,380.07
	S.D. best	1.716e-04	1.767e-04	1.709e-04	1690e-04
	Mean Iter	916	916	979	1101
	Mean CPU	5.493	5.266	17.225	14.552
PSO	Mean best	921,829.41	752,472.88	914,330.41	751,129.29
	S.D. best	2.5116e-04	2.054e-04	2.533e-04	2.855e-04
	Mean Iter	4941	4821	2401	3931
	Mean CPU	11.811	11.420	15.731	17.334

[1] Lambda search is a deterministic approach and each trial produces the same outcome. The results reported here were generated by a single trial at ctol=1.0e-08.

Appendix 20. Minimum Release Constraint Experiment

The numerical results of replicated experiments (ntrials=50) using a (binding) minimum release constraint of 4,000 cfs are reported in this appendix. Some of these results are summarized in graphic form in the main body of the report.

Table 13. Minimum Release Constraint Results

Parameter		168-hour Base Case		Minimum Release Constraint 4,000 cfs (Binding)	
		Summer	Winter	Summer	Winter
LS[1]	Mean best	921,829.41	752,472.91	920,125.14	752,403.53
	S.D. best	na	na	na	na
	Mean Iter	28	28	28	28
	Mean CPU	≤0.002	≤0.002	≤0.002	≤0.002
RCGA	Mean best	921,829.15	752,472.64	919,739.98	752,344.70
	S.D. best	1.831e-03	9.153e-02	1.151e-00	6.765e-02
	Mean Iter	5224	4904	2311	2679
	Mean CPU	9.860	9.258	6.957	5.971
DE	Mean best	921,829.43	752,472.90	920,122.73	752,403.51
	S.D. best	1.716e-04	1.767e-04	1.665e-04	1.588e-04
	Mean Iter	916	916	1271	1277
	Mean CPU	5.493	5.266	13.950	10.325
PSO	Mean best	921,829.41	752,472.88	919,621.85	752,401.83
	S.D. best	2.5116e-04	2.054e-04	3.369e-04	2.633e-04
	Mean Iter	4941	4821	5384	6233
	Mean CPU	11.811	11.420	22.78	19.191

[1] Lambda search is a deterministic approach and each trial produces the same outcome. The results reported here were generated by a single trial at ctol=1.0e-08.

Appendix 21. Program Dictionary

The table below contains the names of the programs used in this analysis and a description of their purpose. This will help facilitate location of the source code file and their reuse at a later date.

Table 14. Program Dictionary

EA	Filename	Purpose
RCGA	RCGEN	Development
RCGA	HDRCGA	Economic dispatch
RCGA	HDRCGAP1	Testing environment
DE	DE04	Development
DE	HDDE	Economic dispatch
DE	HDDEP1	Testing environment
PSO	PSO4	Development
PSO	HDPSO	Economic dispatch
PSO	HDMPSOP1	Testing environment

Mission Statements

The U.S. Department of the Interior protects America's natural resources and heritage, honors our cultures and tribal communities, and supplies the energy to power our future.

The mission of the Bureau of Reclamation is to manage, develop, and protect water and related resources in an environmentally and economically sound manner in the interest of the American public.