

Optimizing a Catalyst: Comparing Generic Particle Swarm Optimization Solver to Lipschitz-continuous Global Optimizer

Thomas Schoene
Department of Computer Science
University of Saskatchewan
110 Science Place
Saskatoon SK S7N 5C9, Saskatchewan, Canada
ths299 at mail.usask.ca

Keywords

global optimization, evolutionary computation, chemical engineering, computer simulation

ABSTRACT

Global optimization deals with finding an optimal solution to a hard problem, that is a problem with multiple minima or maxima. We developed a generic particle swarm optimization solver (G-PSO-S). G-PSO-S addresses global optimization problems and has several advantages over the Lipschitz-continuous global optimizer (LGO). Both, G-PSO-S and LGO, are evaluated on a difficult and expensive global optimization problem, namely optimizing a catalyst.

1. INTRODUCTION

Optimization is the process of finding the best solution, out of many or infinite solutions, that addresses a certain problem. In most cases, this means finding the minimum or the maximum. Global optimization deals with finding an optimal solution to a hard problem, that is a problem with multiple minima or maxima. Finding the best solution to such problems is usually computational infeasible. Therefore, usually just close to optimal solutions are found.

The global optimization problem addressed by this research is the optimization of a catalyst. Catalysts are for example inserted into the gas stream of coal-fired electricity generating stations to reduce the coal plants emission. Catalysts speed up chemical reactions. In this use case, catalysts are used to support the reaction of harmful chemicals into other, less harmful chemicals. The efficiency of catalysts is influenced by parameters such as temperature and pressure. Therefore, optimizing these parameters will improve the system. We are interested in global optimization of such systems in order to have the best possible results.

Three components are essential to solve this global optimization

problem. The first component is a model (simulation) of the catalyst. For given incoming chemical species given a certain configuration, the model will calculate the amount of outgoing chemical species. The second component assigns a fitness score to the output. Domain experts can define a fitness function by specifying which chemical species are desired and which are harmful. For example, it does not make sense to decrease the output of carbon dioxide (CO_2), if by doing so the output of carbon monoxide (CO), formaldehyde (H_2CO), and methane (CH_4) is increased. The third component is a global optimization algorithm, this component finds the input parameter settings for which the fitness is either maximized or minimized. We compared two global optimization techniques based on the quality of the found solution and the elapsed time needed to find that solution. The two methods studied are the widely used commercial global optimization software package Lipschitz-continuous global optimizer (LGO) [Technology 2010], and the promising evolutionary computation global optimization technique particle swarm optimization (PSO).

PSO, as developed by [Kennedy and Eberhart 1995], is an evolutionary computation method. Evolutionary computation methods address the complexity of global optimization problems by finding a relative good solution in a feasible amount of time. They are based on evolutionary mechanisms found in nature [Engelbrecht 2007]. PSO has many variations and its parameters can be tuned to problems, therefore PSO is a versatile global optimization problem solver. PSO is usually applied to continuous problems, but can also solve discrete optimization problems. Even dynamic functions, that is functions that change over time, can be optimized by PSO. Due to this and PSO's good performance, PSO is widely used in industry. For example it is used to solve engineering problems, chemistry problems, and to optimize neural networks.

LGO [Technology 2010] is a commercial optimization tool that can solve global optimization problems. More accurately LGO uses the branch-and-bound global search method (BB), the global adaptive random search (GARS), the multi-start based global random search (MS), or the constrained local search by using a reduced gradient method (LS) to find the global optima. Since four methods can be chosen, LGO is a versatile global optimization solver. Similar to PSO, LGO is widely used in industry.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2010 ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

2. RELATED WORK

[Banga and Seider 1996] showed how stochastic algorithms can be used for global optimization of catalysts. The authors optimized the blend of a tubular reactor using a stochastic algorithm. The paper argues that gradient-based algorithms are not adequate for solving chemical engineering optimization problems such as optimizing a catalyst. This statement is based on the multi-modal and discontinuous objective functions that are found in chemical engineering.

[Holena 2008] used genetic algorithms (GA) to optimize a catalyst. More specifically, they developed a prototype problem driven GA generator. Their prototype is able to generate a genetic algorithm that can address the optimization of catalysts. When optimizing multiple aspects of a catalyst simultaneously, the problem addressed is usually a constrained high dimensional problem. The problem further has a mixture of discrete and continuous variables, and the objective function (fitness function) obtains values empirically. Their optimization targets included optimization of chemical properties and of components preparation methods.

[Huang et al. 2003] used a hybrid genetic algorithm to optimize the design of catalysts. Results showed that the optimized catalysts outperformed previous catalysts.

[Mishra 2006] developed a repulsive particle swarm optimization algorithm (RPSO) in Fortran 77. We implemented our own generic particle swarm optimization solver (G-PSO-S) due to multiple reasons. Firstly, RPSO is implemented in Fortran 77, making it harder to understand and adapt. Secondly, RPSO has all its global optimization test functions in the algorithm's class file. Our philosophy is to define the fitness functions in their own files, making it easy to exchange them at compile time. Further, RPSO does not allow the user to set important parameters such as $c1$, $c2$, and w . We want to use advanced methods such as the dynamic change of $c1$, $c2$, and w , and plan to extend our G-PSO-S with an adaptive particle swarm optimization variation similar to outstanding work by [Zhan et al. 2009]. All those tasks and enhancements fit into the architecture of G-PSO-S, not into RPSO's architecture.

3. PROBLEM STATEMENT

The task is to optimize a catalyst using the model introduced in [Sellers et al. 2009]. The model models the behavior of catalysts. Semi-implicit Runge-Kutta for solving stiff ordinary differential equations [Cash 1976] and Rosenbrock integrators for solving stiff ordinary differential equations [Gottwald and Wanner 1981] are used to solve the model. The problem is treated as a black box, that is it is used as a fitness function without using any known properties except the boundaries. It shall be noted that this research does not further develop the model.

The task is to maximize single species by finding an optimal configuration of the triple temperature ($temp$), carbon dioxide pressure (P_{CO_2}), and methane pressure (P_{CH_4}) for one slide of a palladium (Pd) catalyzed plug flow reactor. t is bounded between 600.0 and 1300.0, P_{CO_2} is bounded between 0.0 and 100000.0, and P_{CH_4} is calculated using Equation (1).

$$P_{CH_4} = 100000.0 - P_{CO_2} \quad (1)$$

The objective function returns a 47 dimensional objective vector (o_v), where the last 14 dimension are the actual result.

The result corresponds to the desorption of certain species, that is the amount in which 15 different species leave the examined catalyst slice. Numbers (dimensions) correspond to species as followed 33 = CH_3COOH , 34 = CH_3OH , 35 = $HCOOH$, 36 = C_2H_2 , 37 = CH_4 , 38 = CO , 39 = C_4H_4 , 40 = C_4H_6 , 41 = C_2H_6 , 42 = C_2H_4 , 43 = CO_2 , 44 = H_2 , 45 = H_2O , 46 = CH_2CO , and 47 = H_2CO .

If Equation (2) holds for the objective vector returned by the model, the result is a valid one.

$$\sum_{i=1}^{32} o_v(i) < 0.4 \quad (2)$$

4. SOLVER

We implemented a generic particle swarm optimization solver (G-PSO-S) in Fortran. Fortran is used in order to compare G-PSO-S to the commercial Lipschitz-continuous global optimization (LGO) solver which is implemented in Fortran.

4.1 Particle Swarm Optimization

Particle swarm optimization (PSO) [Kennedy and Eberhart 1995] is a swarm based global optimization algorithm. It models the behavior of a bird swarm searching for an optimal food source. The movement of a single particle is influenced by its last movement, its knowledge, and the swarm's knowledge. In terms of a bird swarm, this means a bird's next movement is influenced by its current movement, the best food source it ever experienced, and the best food source any bird in the swarm ever experienced. In other words birds (particles) are forced by their last movement, driven by their personal desire, and the swarms movement. PSO's basic equations are:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (3)$$

$$v_{ij}(t+1) = w(t) * v_{ij}(t) + c_1(t)r_{1j}(t)[x_{ijpBest}(t) - x_{ij}(t)] + c_2(t)r_{2j}(t)[x_{jgBest}(t) - x_{ij}(t)] \quad (4)$$

where x represents a particle, i denotes the particle's number, j the dimension, t a time point (iteration), and v is the particle's velocity. x_{pBest} refers to the best location the particle ever visited (the particle's knowledge), and x_{gBest} refers to the best location any particle in the swarm ever visited (the swarm's knowledge). w is the inertia weight and used to weigh the last velocity, c_1 is a variable to weigh the particle's knowledge, and c_2 is a variable to weigh the swarm's knowledge. r_1 and r_2 are random numbers.

All operations in Equations (3) and (4), that is $+$, $-$, and $*$, are implemented as vector operations. If a particle leaves the search space, it is directly "moved back" by using Equation (5), (6), and (7). The first reduction vector can be calculated by using Equation (6), for calculating possible further reduction vectors Equation (7) is used.

$$x_i(t) = x_i(t) - v_{i_back}(n) \quad (5)$$

$$v_{i_back}(1) = v_i(t+1) * rF \quad (6)$$

$$v_{i_back}(n+1) = v_{i_back}(n) * rF \quad (7)$$

Where v_{back} defines the reduction vector, and rF corresponds to the reduction factor that can be set to a value bigger as 0 and smaller as 1 depending on the problem ad-

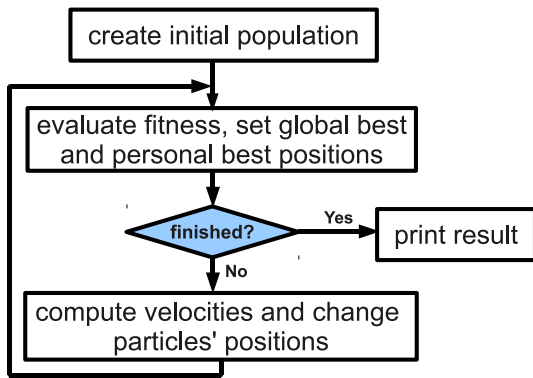


Figure 1: Basic flow of particle swarm optimization.

dressed. n defines the reductions number. x_i , t , and v_i correspond to the variables used in Equation (3) and (4). Figure 1 shows the flow of PSO. First the swarm is initialized. That is, particles and their velocities are filled with random numbers. Now, the particles' fitness is evaluated by calling the fitness function providing the position vectors of the particles. The particle's first fitness is automatically the particle's personal best fitness and the particle's first random position is automatically the particle's personal best position. The global best particle is selected by finding the particle with the highest fitness score in the entire swarm. After those initial steps are finished, Equations (3) and (4) are used to move all particles to their new positions. All fitness scores have to be evaluated again. Personal best positions are updated, if the particles' new fitness scores are better than their personal best scores. The global best particle is updated, if the best particle in the current swarm has a better fitness value as the global best particle. Again, Equations (3) and (4) are used to update all particles' positions. This iterative process continues until a finishing criteria is reached or the maximum number of iterations is exceeded.

In PSO, there is usually a tradeoff between convergence and diversity. To focus on a high diversity means to have the particles far-scattered, searching a large area coarsely. To focus on a high convergence means to have the particles placed close to each other, searching a small area intensively. A promising strategy is to focus on diversity in early iterations while focusing on convergence in late iterations.

Four different PSO variations have been implemented and combined into the generic particle swarm optimization solver (G-PSO-S). Namely global best particle swarm optimization (Equations (3) and (4)), guaranteed convergence particle swarm optimization [Engelbrecht 2007], a modified particle swarm optimizer [Shi and Eberhart 1998], and a particle swarm optimization using the TVAC part (PSO-TVAC) of the self-organizing hierarchical particle swarm optimizer with TVAC (HPSO-TVAC) [Ratnaweera et al. 2004]. The implementation allows for further inclusion of additional variations, if required at latter stages.

Global best particle swarm optimization (G-PSO) is the standard and basic version of PSO.

The global best particle is guaranteed to continue its search within a certain radius if guaranteed convergence particle swarm optimization (GCPSO) is used. Therefore, the global best particle can not get trapped at the current optima.

GCPSO reduces the number of fitness evaluations at single positions. Thus, GCPSO is suited for problems with expensive fitness function evaluations, problems such as the problem addressed here.

Modified particle swarm optimizer (MPSO) [Shi and Eberhart 1998] refers to the idea of decreasing the weight over time. The weight is calculated using Equation (8). The idea behind decreasing the weight over time is to focus on diversity in early stages and to focus on convergence in late stages.

$$w(t) = w_{\text{start}} - (w_{\text{start}} - w_{\text{end}}) * \frac{t}{\text{maxiterations}} \quad (8)$$

where t is the current time point (iteration), maxiterations defines the number of total iterations, w_{start} is the weight meant for early iterations and usually set to 0.9, and w_{end} is the weight meant for late iterations and usually set to 0.4. PSO-TVAC refers to changing c_1 and c_2 over time. Here the weighting factor of the personal knowledge (c_1) and the swarm's knowledge (c_2) are calculated using Equation (9) and (10) respectively. Again, the idea is to have a high diversity in the beginning and decrease it over time in favor of a high convergence.

$$c_1(t) = c_{1\text{start}} - (c_{1\text{start}} - c_{1\text{end}}) * \frac{t}{\text{maxiterations}} \quad (9)$$

$$c_2(t) = c_{2\text{start}} - (c_{2\text{start}} - c_{2\text{end}}) * \frac{t}{\text{maxiterations}} \quad (10)$$

Where t is the current iteration's number (time point), $c_{1\text{start}}$ defines c_1 meant for early iterations and likely set to 2.0, $c_{1\text{end}}$ is c_1 meant for late iterations and likely set to 1.5, $c_{2\text{start}}$ is c_2 meant for early iterations and likely set to 1.6, and $c_{2\text{end}}$ is c_2 meant for late iterations and likely set to 2.0. It is encouraged to use all four variations together, that is using G-PSO, GCPSO, MPSO, and PSO-TVAC. Yet, it is also possible to just use G-PSO, G-PSO with either one of the other three PSO variations, or G-PSO with two of the other three PSO variations.

G-PSO-S, as implemented, can be envisaged as a simple subroutine that has the following input parameters:

- The parameter *maximization* is used to define whether the objective function (fitness function) should be maximized or minimized.
- The parameter w_{range} is a two dimensional array, where the value in the first dimension specifies w_{start} , and the value in the second dimension specifies w_{end} . By setting both values in w_{range} to the same real number, the modified particle swarm optimizer variation is deactivated.
- The parameter $c_{1\text{range}}$ is a two dimensional array, where the value in the first dimension specifies $c_{1\text{start}}$, and the value in the second dimension specifies $c_{1\text{end}}$. By setting both values in $c_{1\text{range}}$ to the same real number, the first half of PSO-TVAC (Equation (9)) is deactivated.
- The parameter $c_{2\text{range}}$ is a two dimensional array, where the value in the first dimension specifies $c_{2\text{start}}$, and the value in the second dimension specifies $c_{2\text{end}}$. By setting both values in $c_{2\text{range}}$ to the same real number, the second half of PSO-TVAC (Equation (10)) is deactivated.

- $n_{\text{particles}}$ specifies the number of particles used, that is it specifies the swarm size.
- $max_{\text{iterations}}$ specifies the maximum number of iterations.
- *reduction factor* specifies rF which is used when particles try to leave the search space. Details on how this parameter is used can be found in Equation (5), (6), and (7).
- *guaranteed convergence* is used to either activate or deactivate GCP SO.
- *radius* specifies GCP SO's radius. That is, the random velocity created by GCP SO will be shortened to *radius*, if bigger than *radius*.
- *dimensions* specifies the number of dimensions a single position (solution/configuration) has.
- *objective dimensions* specifies the number of results (objectives) returned by the objective function (fitness function).
- *objective* specifies the number of the dimension (within *objective dimensions*) that should be optimized. Usually, the fitness function only returns an one dimensional vector and for this case *objective* would be set to 1.

G-PSO-S, as implemented, returns the following parameters:

- *global best value* the fitness score of the global optimum found, that is the fitness score of the solution to the global optimization problem.
- *global best position* the position of the global optimum found, that is the solution to the global optimization problem.

G-PSO-S calls a fitness function. This fitness function is a subroutine that can be exchanged. Therefore, end user of G-PSO-S only need to implement a subroutine that reflects their global optimization problem as well as a program that calls G-PSO-S. The fitness function subroutine and the main program file must be compile together with G-PSO-S, then G-PSO-S can be started and autonomously solves the global optimization problem. G-PSO-S will pass the following parameters to the fitness function.

- *position* defines the (position) vector at which the fitness has to be evaluated by the fitness function. The position vector can be understand as a possible solution to the global optimization problem.
- *maximization* defines whether maximization or minimization is used. This parameter is required as some fitness functions are not well defined in the hole search space. Using this parameter, the fitness function knows whether to return ∞ or $-\infty$, if the fitness function is not defined or incorrect at the requested *position*.

The fitness function will pass the following parameter back to G-PSO-S.

- *fitness* defines the fitness vector (score) at the evaluated position. If the fitness function only has one objective, this vector will only have one dimension. If multiple objectives are combined into one fitness value, again a one dimensional fitness vector will be returned. Yet, also a multidimensional vector, containing multiple objectives, can be returned.

For solving the catalyst problem, the fitness function calls the catalyst model introduced in [Sellers et al. 2009]. The fitness function receives a two dimensional particle's *position* vector. The first value defines *temp* and the second value P_{CO_2} . P_{CH_4} is calculated by using Equation (1). Using this three values, the catalyst model is called. The model returns a 47 dimensional objective vector (o_v), where the last 15 dimension are the actual result. If Equation (2) holds for the vector returned by the model, the vector represents a valid fitness score that can be returned directly. If it does not hold and maximization is used, values $o_v(33)$ to $o_v(47)$ will be set to $-\infty$. If it does not hold and minimization is used, values $o_v(33)$ to $o_v(47)$ will be set to ∞ . This forces G-PSO-S's particles out of undefined regions.

4.2 Lipschitz-continuous Global Optimization

Lipschitz-continuous global optimization (LGO) solver is a commercial global optimization problem solving environment [Technology 2010] implemented in Fortran. LGO is widely used in industry. In LGO, the methods branch-and-bound global search method (BB), global adaptive random search (GARS), multi-start based global random search (MS), and constrained local search by using a reduced gradient method (LS) can be chosen as algorithm for solving the global optimization problem addressed [Pinter 2005]. As LGO is a commercial product and implementation details are not published, a detailed description of the implementation and the used methods is not possible here.

The LGO used is the command line license, not the graphical user interface (GUI) license. In order to solve a global optimization problem with the LGO command line tool, the following steps have to be carried out:

1. A configuration file which sets parameters such as search space (bounds), maximum number of fitness evaluations, and maximum run time for LGO has to be created. The configuration file is named LGO.IN.
2. A Fortran subroutine which returns the fitness value to a given solution to the optimization problem must be implemented. This function is similar to the fitness function used for G-PSO-S. LGO's fitness function file is usually called USERFCT.FOR.
3. LGO's program file (LGO.o), the fitness function file (USERFCT.FOR), and any other required Fortran files must be compiled. The resulting program can be used and will return the optimization results into a file called LGO.SUM.

As this process is quite time consuming, we developed a Python script that for optimizing different species, automatically edits the LGO.IN and USERFCT.FOR files. After the files are changed, the script compiles LGO, then runs LGO, and finally parses the solution file LGO.SUM.

5. EVALUATION

For the Lipschitz-continuous Global Optimization solver (LGO), the following parameter settings have been used. The number of variables is set to 2, and the number of constraints is set to 0. For $temp$, the lower bound has been set to 600., the upper bound to 1300., and the nominal solution to 950.. For P_{CO_2} , the lower bound is set to 0., its upper bound to 100000., and its nominal solution to 50000.. It shall be noted that the full stop at the end of floating point numbers is required, otherwise LGO throws runtime errors. The maximum number of function evaluations in the global search phase has been set to 200, and the maximum number of function evaluation in the global search phase without improvement has been set to 100. The target objective value in the global and local search phase has been set to $-1.79e+32$. The Merit function precision improvement threshold in the local search phase as well as the Kuhn-Tucker local optimality conditions tolerance in the local search phase have been both set to $1.e-12$. The random seed number is set to 27, and the maximum execution time is set to 1200 seconds. The operational mode is set to an number between 0 and 4 depending on which LGO variation is tested. Where 0 means LGO will use local search by using a reduced gradient method (LS), 1 means LGO will use a branch-and-bound global search method (BB) with an up-following LS, 2 means LGO will use a global adaptive random search (GARS) with an up-following LS, and 3 means LGO will use a multi-start based global random search (MS) with an up-following LS. For the generic particle swarm optimization solver (G-PSO-S), following parameter settings have been used. The first parameter, that is *maximization* is set, that is maximization is used. w_{range} is set to (0.95,0.45), therefore w_{start} is 0.95 and w_{end} is 0.45. $c1_{range}$ is set to (2.0,1.0), thus $c1_{start}$ is 2.0 and $c1_{end}$ is 1.0. The parameter $c2_{range}$ is set to (1.95,2.0), therefore $c2_{start}$ is 1.95 and $c2_{end}$ is 2.0. $n_{particles}$ is set to 22 and $max_{iterations}$ is set to 21, thus the number of function evaluations (FE) is 484. G-PSO-S's number of function evaluations have been set to a similar number as used by LGO, not to a number that yields the best results. *reduction factor* (rF) is set to 0.5 and the *radius* is set to 100.0. *dimensions* is set to 2, *objective dimensions* is set to 47, and *objective* is set to a value between 33 and 47, depending on the objective that should be maximized. *guaranteed convergence* is set (activated) for the test cases addressed by "4 variations" in Tables (1) and (2) and deactivated for the test cases addressed by "3 variations" in Tables (1) and (2). More specifically, "4 variations" refers to G-PSO-S using a combination of global best particle swarm optimization (G-PSO) [Kennedy and Eberhart 1995], guaranteed convergence particle swarm optimization (GCP SO) [Engelbrecht 2007], a modified particle swarm optimizer (MPSO) [Shi and Eberhart 1998], and a particle swarm optimization with TVAC (PSO-TVAC) [Ratnaweera et al. 2004], and "3 variations" refers to G-PSO-S using a combination of G-PSO, MPSO, and PSO-TVAC. For the "3 variations" test cases $max_{iterations}$ is changed to 10, and $n_{particles}$ is changed to 11, therefore 121 function evaluations are used. The results given by the "3 variations" G-PSO-S shall be compared to LGO's LS and BB+LS methods that use lesser function evaluations but show worse results. "4 variations" G-PSO-S shall be compared to LGO's GARS+LS and MS+LS methods that use more function evaluations and show better results.

As objective 33 showed a unique behavior, only for that objective's "4 variations" test case $n_{particles}$ has been changed from 22 to 30 and $max_{iterations}$ has been changed from 21 to 14. For the "3 variations" objective 33 test case, $n_{particles}$ has been changed to 15 and $max_{iterations}$ has been changed to 7.

Tables (1) and (2) show the results when maximizing all possible single objectives (species). The column species holds the number of species (objective) that shall be optimized, column solver refers to the solver used, column type identifies the solver variation used, column maximum holds the fitness score of the maximum (optimum) found, and column FE identifies the number of function evaluations used to find that maximum. In column type, LS identifies that LGO's LS was used (mode 0), BB+LS identifies that LGO's BB with an following LS was used (mode 1), GARS+LS identifies that LGO's GARS with an following LS was used (mode 2), MS+LS identifies that LGO's MS with an following LS was used (mode 3), 3 variations identifies that G-PSO-S with G-PSO, MPSO, and PSO-TVAC was used, and 4 variations identifies that G-PSO-S with G-PSO, GCP SO, MPSO, and PSO-TVAC was used. The winning method of the comparison between G-PSO-S's "4 variations", LGO's GARS+LS, and LGO's MS+LS is highlighted in **blue**, and the winning method of the comparison between G-PSO-S's "3 variations", LGO's LS, and LGO's BB+LS is highlighted in **green**. Winning means having a higher maximum objective value. Function evaluations are only considered if two methods have exactly the same maximum objective value, in this case the method using less function evaluations wins.

5.1 Further considerations

In this section, shortcomings of LGO that are successfully addressed by the generic particle swarm optimization solver (G-PSO-S) are described.

LGO's code can not be extended as the distribution only includes the object file, not the source code. As we developed G-PSO-S, the source code can be extended.

LGO does not support multi-objective optimization as a means of finding the parameter front. While currently our G-PSO-S does not support multi-objective optimization as a means of finding the parameter front, it would not be difficult to extend G-PSO-S for multi-objective optimization. Optimization of discrete problems is not supported by LGO. Particle swarm optimization (PSO), if adapted, can solve discrete optimization problems. In a other project, we successfully addressed a match making problem implementing a discrete PSO using Java.

LGO can not solve dynamic global optimization problems, that is global optimization problems which change their behavior, that is their fitness functions shape, over time. G-PSO-S can seamlessly be enhanced to address dynamic global optimization problems.

In LGO, only minimization is implemented. For maximization, the fitness function must multiply each fitness value with -1 before returning it. Doing so the solution's fitness value has to be multiplied with -1 again. G-PSO-S supports both, maximization and minimization. Therefore, no change to the fitness function and no correction to the solution's fitness value must be applied when doing maximization. The user can choose between maximization and minimization by using G-PSO-S's input parameter *maximization*.

Table 1: Comparison of LGO to G-PSO-S objectives (species) 33 to 40.

Species	Solver	Type	Maximum	FE
33	LGO	LS	0.0001347606	155
		BB+LS	0.0001347606	334
		GARS+LS	0.0023806758	473
		MS+LS	0.0023806750	473
33	PSO	3 variations	0.0005755203	120
		4 variations	0.0023794196	480
34	LGO	LS	65.28235812	144
		BB+LS	65.28260307	328
		GARS+LS	768.51283742	390
		MS+LS	768.51283742	390
34	PSO	3 variations	648.54407391	121
		4 variations	768.51283305	484
35	LGO	LS	8711.91114	190
		BB+LS	8711.91889	364
		GARS+LS	382285.50961	498
		MS+LS	382285.50961	498
35	PSO	3 variations	371309.17065	121
		4 variations	382285.42835	484
36	LGO	LS	4679879938	145
		BB+LS	296672160418	321
		GARS+LS	296672160418	345
		MS+LS	296672160418	345
36	PSO	3 variations	295414289304	121
		4 variations	296672154571	484
37	LGO	LS	44033033955075	194
		BB+LS	44033045681634	323
		GARS+LS	44033045681634	392
		MS+LS	44033045681634	392
37	PSO	3 variations	49956630033567	121
		4 variations	51847019955634	484
38	LGO	LS	249395424483	100
		BB+LS	249395424483	288
		GARS+LS	249395424483	383
		MS+LS	249395424483	383
38	PSO	3 variations	249133808976	121
		4 variations	249395250806	484
39	LGO	LS	660078094.4	132
		BB+LS	660078094.4	320
		GARS+LS	693174560.4	435
		MS+LS	693174560.4	435
39	PSO	3 variations	1074382818.3	121
		4 variations	1074396262.6	484
40	LGO	LS	121.05829684	115
		BB+LS	121.05829684	298
		GARS+LS	121.05829684	420
		MS+LS	121.05829684	420
40	PSO	3 variations	120.28890482	121
		4 variations	121.05588660	484

Table 2: Comparison of LGO to G-PSO-S objectives (species) 41 to 47.

Species	Solver	Type	Maximum	FE
41	LGO	LS	23232958.684	116
		BB+LS	23232958.684	269
		GARS+LS	23232958.684	399
		MS+LS	23232958.684	399
41	PSO	3 variations	23227786.944	121
		4 variations	23232851.210	484
42	LGO	LS	25892073696.3	103
		BB+LS	41285927835.7	294
		GARS+LS	55270802390.9	356
		MS+LS	55270802390.9	356
42	PSO	3 variations	54841363184.7	121
		4 variations	55270802120.1	484
43	LGO	LS	30875088419519	196
		BB+LS	30875103850260	291
		GARS+LS	30875103850260	352
		MS+LS	30875103850260	352
43	PSO	3 variations	32097460847863	121
		4 variations	32135484648632	484
44	LGO	LS	4533730	127
		BB+LS	972594174910	288
		GARS+LS	972594174910	323
		MS+LS	972594174910	323
44	PSO	3 variations	972536021341	121
		4 variations	972594162093	484
45	LGO	LS	153854166578	153
		BB+LS	153854169288	336
		GARS+LS	157718463079	454
		MS+LS	157718463079	454
45	PSO	3 variations	163023542012	121
		4 variations	163023593569	484
46	LGO	LS	202654.85629	271
		BB+LS	202654.85628	453
		GARS+LS	203082.15431	673
		MS+LS	203082.15431	673
46	PSO	3 variations	208402.40370	121
		4 variations	208506.47103	484
47	LGO	LS	189220707.31	85
		BB+LS	189220707.31	280
		GARS+LS	189220707.31	361
		MS+LS	189220707.31	361
47	PSO	3 variations	188562269.58	121
		4 variations	189218434.06	484

LGO encrypts solutions with an absolute value of 100000000 (10⁹) and bigger into "*****" in its solution file. It is impossible to come from "*****" back to the solution as all numbers above 1e⁹ are encoded into the same amount of stars. This means that user have to call the fitness function by hand, using the found configuration (position in the search space) which is provided in the solution file. For a commercial product this is bad, especially when considering that it probably just means changing a "f10.10" to a "f50.10" in LGO's Fortran code. Using G-PSO-S, solutions do not have to be reevaluated to find their fitness values.

LGO does not support parallelism for solving a global optimization problem. G-PSO-S can be extended to support parallelism.

LGO does not provide a input configuration file specification. Only a manual [Pinter 2005] is shipped with the LGO command line license. G-PSO-S does not have such problems as its interface will be specified soon and its code can be read.

LGO's parsing of the configuration file is very inflexible and its behavior is often off-key. For example setting a float parameter to 1. works, but if 1.0 is provided a runtime error is thrown. As those runtime errors do not identify in which line of the configuration file the error occurs, figuring out such mistakes can be challenging. Another issue is that space characters have a meaning. If comments start one position to early, run time errors are thrown as well. Furthermore, the ordering of the parameters specified is important. Unlike in nice and clear Java like configuration files where parameter keys are used, LGO does not use any parameter keys. A more modern configuration file style by using a flat file with simple <key>=<value> entries, would make LGO's configuration faster and easier. G-PSO-S receives all its parameters by the subroutine call, thus no difficult and badly specified configuration file has to be used. If users want to use a configuration file, they can write their own code reading a configuration file and calling the G-PSO-S with the read values.

LGO does not follow the maximum function evaluations specified in its configuration file. Thus making it nearly impossible to compare LGO to other global optimization problem solvers when number of function evaluations are strictly bounded. G-PSO-S allows the exact specification of the number of function evaluations. G-PSO-S's number of function evaluations can be calculated by Equation (11).

$$FE = (max_{iterations} + 1) * n_{particles} \quad (11)$$

Where FE corresponds to the number of fitness evaluations used. And $n_{particles}$ and $max_{iterations}$ correspond to the homonymous G-PSO-S parameters.

LGO as used, is limited to just 1000 variables. That is it can not address problems with more than 1000 dimensions. G-PSO-S is only limited by Fortran's and the used computer's limitations, thus it can address much higher dimensional problems.

LGO is a expensive commercial product. Developing and using G-PSO-S is relatively cheap.

Mathematical terms such as nominal value, are miss-used in the input parameter file. Users have to specify an upper bound, a lower bound and a nominal value between those bounds in the configuration file. Yet, the upper and lower bounds are already nominal values, so specifying additional

possible nominal solutions does not seem to make sense. This value should be rather called likely solution, otherwise the upper - and the lower bound would have to be renamed to nominal upper - and nominal lower bound. The use of the term constraint is also confusing. That is it is not clear how constraints are assigned to variables. The philosophy of G-PSO-S is to let the user check and handle constraints within the fitness function, that is if one constraint is violated, the fitness function should return $-\infty$ if maximization is used and ∞ if minimization is used.

6. CONCLUSION

The generic particle swarm optimization solver (G-PSO-S) showed surprisingly good results, keeping up with the commercial Lipschitz-continuous global optimizer (LGO). For optimizing certain species, G-PSO-S outperforms LGO (37, 39, ...), while for other species LGO outperforms G-PSO-S (36, 38, ...). This shows that the no free lunch (NFL) theorem for optimization [Wolpert and Macready 1997] still holds. Both solvers tend to find a solution in the same region, suggesting that both find the global optima.

Considering just the comparison between G-PSO-S's 3 variations, LGO's LS, and LGO's BB+LS which is measuring the performance of the solvers considering global optimization problems with strictly bounded number of function evaluations, G-PSO-S is outperforming LGO 9 times, and LGO outperforms G-PSO-S 6 times. In the cases where G-PSO-S outperforms LGO, the largest improvement is a maximum that is 42.6 times higher (objective 35). In the cases where LGO outperforms G-PSO-S, the largest improvement is a maximum that has an improvement in its third highest digit (objective 36). This suggests that G-PSO-S is able to find global optima faster and should be used for time critical and very large and difficult global optimization problems.

Considering just the comparison between G-PSO-S's 4 variations, LGO's GARS+LS, and LGO's MS+LS which is measuring the performance of the solvers considering global optimization problems with roughly bounded number of function evaluations, G-PSO-S is outperforming LGO 5 times, and LGO outperforms G-PSO-S 10 times. In the cases where G-PSO-S outperforms LGO, the largest improvement is a maximum that is 1.55 times higher (objective 39). In the cases where LGO outperforms G-PSO-S, the largest improvement is a maximum that has an improvement in its fourth highest digit (objective 33). This suggests that G-PSO-S is more reliable in finding the global maximum. It further suggests that LGO explores the region of the global optimum more accurately. This suggests that G-PSO-S should use a local search after finding the global optimum in order to get as accurate results as LGO.

G-PSO-S showed better results after its parameters were tuned to the problem. This tuning was done empirically. Empirical tuning of all parameters is time consuming and often does not find the optimal configuration, suggesting that adaptive particle swarm optimization (APSO) [Zhan et al. 2009] should be included into G-PSO-S .

High evaluation times (above 8 hours) for complete evaluations of LGO and G-PSO-S suggest that G-PSO-S should be parallelized to be able to address realistic 10000 slice Pd catalyst problems.

When considering LGO's shortcomings, G-PSO-S's reliability in finding the global optima, and G-PSO-S's good performance under strictly bounded number of function evalu-

ations, G-PSO-S is the better choice for global optimization of the catalyst problem.

It shall be noted that due to the nature of computer simulations, such as the catalyst model used, obtained results always have to be verified by actual experiments.

7. FUTURE WORK

Local best particle swarm optimization (L-PSO) is currently implemented as an additional particle swarm optimization (PSO) variation for the generic particle swarm optimization solver (G-PSO-S).

Local search will be added to G-PSO-S, that is local search will be used on the found global optimum. This hybrid G-PSO-S local search (G-PSO-S LS) method is expected to yield more accurate results compared to G-PSO-S, without adding significantly more function evaluations.

An adaptive change of at least w , $c1$, and $c2$ based on the algorithms state information such as swarm's overall crowding distance and relative change in the global best particle's position will be implemented, roughly following ideas of [Zhan et al. 2009]. Adaptive G-PSO-S (AG-PSO-S) is expected to outperform G-PSO-S.

AG-PSO-S will be parallelized. For global optimization problems with expensive fitness function evaluations, the fitness function evaluations of AG-PSO-S will be parallelized, following ideas of [Schutte et al. 2004]. For global optimization problems with inexpensive fitness function evaluations, sub-swarms will be parallelized, following ideas of [Niu et al. 2007]. Both architectures will improve the performance as more fitness function evaluations can be computed in the same amount of time. The parallel AG-PSO-S (PAG-PSO-S) is expected to outperform all, the Lipschitz-continuous global optimizer (LGO), G-PSO-S, and AG-PSO-S, significantly.

A fitness function which combines all objectives into one by specifying which chemical species are desired and which are harmful has to be specified. The global optimization of this function will find a close to optimal parameter configuration for Pd catalysts.

Using PAG-PSO-S, the catalyst problem shall be solved for a realistic 10000 slices Pd catalyst problem.

8. ACKNOWLEDGEMENT

We would like to thank Raymond J. Spiteri, Simone Ludwig, and Andrew Kroshko for excellent support during the project. We would also like to thank Ian McQuillan, Simone Ludwig, and Raymond J. Spiteri for agreeing to review and grade this paper. Special thanks to Tony Kusalik and Michael C. Horsch for guidance and support throughout the CMPT 880 course, which was leading to this paper.

9. REFERENCES

- [Banga and Seider 1996] BANGA, J. R. AND SEIDER, W. D. 1996. Global optimization of chemical processes using stochastic algorithms.
- [Cash 1976] CASH, J. R. 1976. Semi-implicit Runge-Kutta procedures with error estimates for the numerical integration of stiff systems of ordinary differential equations. *23*, 3, 455–460.
- [Engelbrecht 2007] ENGELBRECHT, A. 2007. *Computational Intelligence - An Introduction 2nd Edition*. Wiley.
- [Gottwald and Wanner 1981] GOTTWALD, B. A. AND WANNER, G. 1981. A reliable Rosenbrock integrator for stiff differential equations. *26*, 4, 355–360.
- [Holena 2008] HOLENA, M. 2008. Genetic algorithms for the optimization of catalysts in chemical engineering. In *Proceedings of the World Congress on Engineering and Computer Science*. San Francisco, USA.
- [Huang et al. 2003] HUANG, K., ZHAN, X.-L., CHEN, F.-Q., AND LÁJ, D.-W. 2003. Catalyst design for methane oxidative coupling by using artificial neural network and hybrid genetic algorithm. *Chemical Engineering Science* *58*, 1, 81 – 87.
- [Kennedy and Eberhart 1995] KENNEDY, J. AND EBERHART, R. 1995. Particle swarm optimization. *IEEE International Conference on Neural Networks - Conference Proceedings* *4*, 1942 – 1948.
- [Mishra 2006] MISHRA, S. 2006. Global optimization by particle swarm method: a fortran program. MPRA Paper 874, University Library of Munich, Germany. Aug.
- [Niu et al. 2007] NIU, B., ZHU, Y., HE, X., AND WU, H. 2007. Mcps: A multi-swarm cooperative particle swarm optimizer. *Applied Mathematics and Computation* *185*, 2, 1050 – 62.
- [Pinter 2005] PINTER, J. D. 2005. *LGO - A Model Development and Solver System for Continuous Global Optimization User Guide*. Pinter Consulting Services.
- [Ratnaweera et al. 2004] RATNAWEERA, A., HALGAMUGE, S. K., AND WATSON, H. C. 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* *8*, 3, 240 – 255.
- [Schutte et al. 2004] SCHUTTE, J., REINBOLT, J., FREGLY, B., HAFTKA, R., AND GEORGE, A. 2004. Parallel global optimization with the particle swarm algorithm. *International Journal for Numerical Methods in Engineering* *61*, 13, 2296–315.
- [Sellers et al. 2009] SELLERS, H., SPITERI, R. J., AND PERRONE, M. 2009. $CO_2 + CH_4$ chemistry over Pd : Results of kinetic simulations relevant to environmental issues. *113*, 6, 2340–2346.
- [Shi and Eberhart 1998] SHI, Y. AND EBERHART, R. 1998. A modified particle swarm optimizer. In *Proceedings of the IEEE World Congress on Computational Intelligence*. 69 – 73.
- [Technology 2010] TECHNOLOGY, P. D. 2010. <http://www.aimms.com/features/solvers/lgo>.
- [Wolpert and Macready 1997] WOLPERT, D. H. AND MACREADY, W. G. 1997. No free lunch theorems for optimization.
- [Zhan et al. 2009] ZHAN, Z.-H., ZHANG, J., LI, Y., AND CHUNG, H. S.-H. 2009. Adaptive particle swarm optimization. *Trans. Sys. Man Cyber. Part B* *39*, 6, 1362–1381.