CSE 620 Final Project: Bifunctional Catalyst

Sima Shafaei
Student, CSE 620
University of Louisville
Louisville, KY
sima.shafaei@louisville.edu

Jim LeFevre
Student, CSE 620
University of Louisville
Louisville, KY
james.lefevre@louisville.edu

***Abstract*—One application of optimization strategies is in chemical engineering. A particular problem is the production of benzene from methylcyclopentane. This reaction involving seven chemical species can be catalyzed by a blend of two compounds. The exact proportion of these two compounds at any given time affects the rate of the reaction. We applied iterative dynamic programming and a genetic algorithm with deterministic crowding to this problem. Deterministic crowding appears to be the more useful algorithm for this problem, because it was able to locate the global peak and other peaks in a relatively short time.**

I. Original Proposal

Our idea for a real-world optimization problem is the conversion of methylcyclopentane to benzene. This is a 7-stage process. Two catalysts are used that have different effects in each stage. The performance of a catalyst mixture can be modeled by a set of differential equations. Previous research found at least 300 local optima.

II. Introduction

A process for converting methylcyclopentane to benzene makes use of a bifunctional catalyst blend to speed up the reaction. The catalyst blend contains a hydrogenation component and an isomerization component. The hydrogenation proportion u can be varied from 0.60 to 0.90. The process can be subdivided into 10 stages, with 10 corresponding u values. What 10 values of u_i will yield the greatest amount of benzene? It may be possible to use heuristic methods to find a reasonably good solution in much less time than it would take to find the u_i values that provide the exact optimum. Table I lists previous studies of this problem.

TABLE I. Previous Optimization Studies of the Bifunctional Catalyst Problem

Reference	Name of Algorithm	Category	Known Complexity
[1]	Nonlinear Programming	Total	Unable to determine
[1]	Dynamic Programming	Total (subproblems); approximate	$O(M*N*i)$
[2]	Dynamic Programming	Total (subproblems); approximate	$O(M*i)$ ($N = 1$)
[3]	Branch and bound	Total; exact	Unable to determine

III. Optimization Problem

Seven chemical species are involved in the reaction, which is diagrammed in Figure 1.

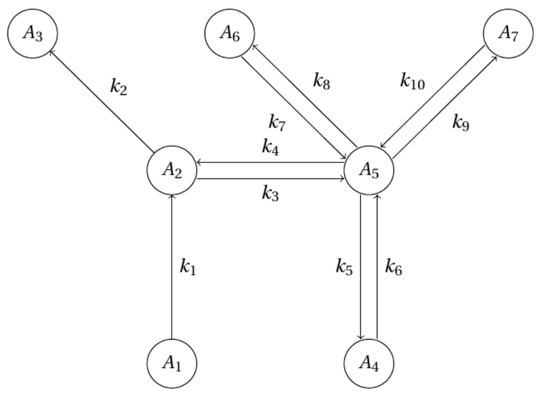


Fig. 1. Reaction diagram. A_1 represents methylcyclopentae, A_7 represents benzene, and the other nodes represent intermediates and byproducts.

Using empirical data, the rate of change of the quantity of each chemical can be expressed as a derivative with respect to time. The set of 7 differential equations can then be solved to determine the final quantities of each chemical. The equations are:

$$\frac{dx_1}{dt} = -k_1x_1 \quad (1)$$

$$\frac{dx_2}{dt} = k_1x_1 + (k_2 + k_3)x_2 + k_4x_5 \quad (2)$$

$$\frac{dx_3}{dt} = k_2x_2 \quad (3)$$

$$\frac{dx_4}{dt} = -k_6x_4 + k_5x_5 \quad (4)$$

$$\frac{dx_5}{dt} = k_3x_2 + k_6x_4 - (k_4 + k_5 + k_8 + k_9)x_5 + k_7x_6 + k_{10}x_7 \quad (5)$$

$$\frac{dx_6}{dt} = k_8x_5 - k_7x_6 \quad (6)$$

$$\frac{dx_7}{dt} = k_9x_5 - k_{10}x_7 \quad (7)$$

The value of each constant $k_1...k_{10}$ is a function of both the stage and the value of u_i at that stage. In reality the value of u could be adjusted continuously, but to make the problem feasible all previous studies have discretized the value of u to 10 stages. Time is measured in terms of the mass of catalyst used. Each stage uses 200 grams (assuming the reactor processes 1 mole of methylcyclopentane per hour).

The 7-element vector representing the quantity of each chemical at a given point in time, $\mathbf{x}$, is initially set to 1.0 for methylcyclopentane and 0.0 for all other chemicals.

The objective is to maximize the amount of benze present at the end of the reaction. The input is $\mathbf{u}$, the set of 10 hydrogenation fraction values, one for each stage of the reaction. The values of $\boldsymbol{u}$ cannot exceed the range [0.60,0.90]. In addition, no chemical quantity can exceed the range [0,1], but this is not expected to occur given the starting conditions and empirically determined *k* values. Briefly, we can say that, this bifunctional catalyst optimization problem is a nonlinear and non-convex constrained maximization of a numeric problem with a large number of local optima. Table II expresses the characteristics of this optimization problem.

TABLE II. OPTIMIZATION PROBLEM FORMULATION

Cost Function	$x_7(2000)$
Variable	$\mathbf{u} = [u_1,\ldots,u_{10}]$
Constraints	$u_i >= 0.60$ for $i=1,\ldots,10$.
	$u_i <= 0.90$ for $i = 1,\ldots,10$

A. *Search Space*

The search space has 10 identical dimensions. Each axis is a real number with a range [0.60, 0.90]. Real numbers are expressed with 8 significant digits.

B. *Difficulties*

The fitness landscape is known to exhibit over 300 local maxima [3]. Therefore, many algorithms are at risk of converging to a local maximum.

Computational complexity is also a consideration. The high dimensionality of the search space increases computational complexity. The fitness function has a high computational complexity because it requires the solution of a system of differential equations.

C. *Related Problems*

In [1] this problem is formulated as an NLP classical problem. To provide the necessary gradients, they used a simple method which uses finite differences. The performance index $I = x_7(\mathrm{tf})$ can be written as an explicit function of the controls $u(0), \ldots, u(9)$,

$$I = I[u(0), u(1), \ldots, u(9)]$$

So that the gradient can be obtained by simply perturbing the control,

$$\frac{\partial I}{\partial u(j)} \approx (I[u(0),\ldots,u(j)+\delta,\ldots,u(9)] - I[u(0),\ldots,u(9)])/\delta$$

If δ is sufficiently small (e.g. 10^{-6}) the approximation is very good for use in NLP.

The problem description can be transformed into the following formal NLP statement:

$$\text{maximize } f(\mathbf{u})$$
$$\text{subject to } g_i(\mathbf{u}) \geq 0, i = 1,\ldots,20$$
where

$$f(\mathbf{u}) = \mathbf{x}(2000)^t * [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^t$$
$$\mathbf{x}(t) = \mathbf{x}_0 + \int_0^t \frac{d\mathbf{x}}{dt} dt$$
$$\mathbf{x}_0 = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$
$$\frac{d\mathbf{x}}{dt} = \left[\frac{dx_1}{dt} \quad \cdots \quad \frac{dx_7}{dt}\right]^t$$
$$g_1(\mathbf{u}) = \mathbf{u}^t * [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^t - 0.60$$
$$g_2(\mathbf{u}) = \mathbf{u}^t * [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^t - 0.60$$
$$\ldots$$
$$g_{11}(\mathbf{u}) = -(\mathbf{u}^t * [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^t - 0.90)$$
$$\ldots$$
$$g_{20}(\mathbf{u}) = -(\mathbf{u}^t * [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^t - 0.90)$$

IV. OPTIMIZATION METHODS

Dynamic programming has been used successfully on this problem in the past [1][2], so we chose this as a classical method. Specifically, we used iterative dynamic programming (IDP), in which each iteration searches a space that is (1) centered on the result of the prior iteration, and (2) reduced in size from the search space of the prior iteration. For a meta-heuristic, we evaluated a basic genetic algorithm (GA), a GA with sharing, and a GA with deterministic crowding (DC). Of these, DC exhibited the best performance in preliminary experiments, and was selected for further analysis.

A. *Algorithms' Pseudocode*

1) DC

The initial phenotypes were assigned by random assignment from a uniform distribution across the search space. No genotype encoding was used; phenotypes were directly altered for crossover and mutation.

DC pseudocode follows, in which *g* is the number of iterations/generations, *n* is the population size, *d*(*a*, *b*) returns the distance between individuals *a* and *b*, and the variation operator includes both crossover and mutation [4]:

```
for $i$ in 1:$g$
    for $j$ in 1:$n/2$
        Randomly select (without replacement) parents $p_1$ and $p_2$
        Apply variation operator on $p_1$ and $p_2$ to generate $c_1$ and $c_2$
        Evaluate fitness $f$ of $c_1$ and $c_2$
        if $d(p_1, c_1) + d(p_2, c_2) <= d(p_1, c_2) + d(p_2, c_1)$
            if $f(c_1) > f(p_1)$, then select $c_1$ and remove $p_1$
            if $f(c_2) > f(p_2)$, then select $c_2$ and remove $p_2$
        else
            if $f(c_2) > f(p_1)$, then select $c_2$ and remove $p_1$
            if $f(c_1) > f(p_2)$, then select $c_1$ and remove $p_2$
```

The fitness function is the value of x_7 at $t = 2000$.

In the variation operator, crossover is implemented as follows:

$$c_1 = \alpha p_1 + (1 - \alpha)p_2$$

$$c_2 = \alpha p_2 + (1 - \alpha)p_1$$

α is a random number from a uniform distribution in [$-\gamma$, $1 + \gamma$], where γ is an input parameter.

Mutation is implemented as follows:

$$c_1' = c_1 + \sigma\alpha$$

$\sigma = 0.1$ and α is a random number from a normal distribution with mean 0 and standard deviation 1

In addition to g, n, γ, and σ, DC accepts the following parameters:

- p_m: probability that a child will mutate
- p_c: probability that crossover will occur when a pair of parents reproduces

The output of DC is a vector of n solutions, each having a phenotype (**u**) and fitness (x_7 at $t = 2000$).

2) IDP

The following IDP algorithm is quoted from [1]:

"1. Choose the number of x-grid points N and the number of allowable values M for the control $u(k)$

2. Choose an initial control policy $u(0)$, $u(1)$, …, $u(9)$.

3. Choose an initial region $r(k)$, k=0,1,2, …,9

4. By choosing N values of control evenly distributed inside the allowable region about the control policy, integrate Equation (1) – (7) N times with the given initial condition $x(0)$=[1 0 0 0 0 0 0]T from time 0 to t_f, to generate N values for the x-grid at each time stage.

5. Starting at stage 10, corresponding to time 1800 g . h/mol for each x-grid point integrate Equation (1) – (7) from 1800 to 2000 once with each of M allowable values of $u(9)$. For each grid point choose amongst the M values used, the control $u(9)$ that gives the greatest value to the performance index, i.e., the highest value of $x7$ at time 2000 g . h/mol. Store the value of control $u(9)$ for use in step 6.

6. Step back to stage 9, corresponding to time 1600 g . h/mol and integrate Equation (1)-(7) from time 1600 to 1800 for each x-grid point once with each the M allowable values of control $u(9)$ from step 5 that corresponds to the grid point closest to the resulting x at time 1800 g . h/mol and continue integration from 1800 to 2000 to give M values of performance index to compare. Store the best value of control $u(8)$ that yields the maximum value of the performance index.

7. Repeat this procedure for stages 8,7,6, etc. until stage 1, corresponding to the initial time t=0 and initial condition is reached. This grid has only the single point. Now assign M values to $u(0)$ and integrate Equations (4)-(10) from t=0 up to t=200 g .h/mol. At t= 200 g. h/mol use the control policy that has been sorted from the previous step, that corresponds to the grid point at stage 2 that is closest to the state at t=200 and proceed in this manner until we reach t= 2000 g .h/mol. There are now M values of the performance index to compare. And we choose the set $u(0)$, $u(1)$, …, $u(9)$ that yields the maximum value, and store this set.

8. Reduce the region for the allowable control

$r(j+1)(k) = \gamma\ r(j)(k)$, k=0,1,…,9

where j is the iteration index and the contraction factor γ is less than 1, such as 0.70.

9. Choose the control policy $u(0)$, $u(1)$, …, $u(9)$ from step 7, increment j by 1 and go to step 4. Continue the procedure for a specific number of iterations such as 20 and examine the results."

To summarize, the input parameters for IDP are:

- N: number of grid points per stage (each grid point being a starting point for integration, with grid points evenly spaced across the current search space)
- M: number of u values considered per grid point (the grid point will be integrated with M different random u values, and the u value that yields the best fitness is stored for possible later use)
- r: width of the search space in the first iteration
- γ: factor by which search space shrinks with each iteration
- Initial u_p: the u value that is used for the preliminary integration that generates the grid; this initial value is a scalar that is applied for each of the 10 stages, although each stage may have a different value after the first iteration
- i: iterations, the number of times the IDP restarts with a new u_p and smaller r

The output of IDP is a single solution u with fitness x_7 at $t = 2000$.

B. *Algorithms' Relation to Existing Methods*

DC evaluates complete solutions [5]. Evaluation always considers a full set of 10 u_i values. The solutions are approximate: the individual with the maximum fitness may not represent an exact maximum. Refinement of DC solutions using hill-climbing can help to find a more exact peak value. Aside from that, the peaks identified by DC do not necessarily include the global maximum.

IDP operates by determining solutions to reduced problems. The problem is reduced to the level of a single stage. Multiple candidate solutions are generated and evaluated for each stage and the most fit solution is retained. Complete solutions are built from a chain of subproblem solutions. The solutions are approximate in that each iteration returns the best solution out of the solutions it explored, but there is no guarantee that any iteration returns the global maximum, or even a local maximum.

TABLE III. SUMMARY OF RELATION TO EXISTING METHODS

	Total/Partial	Exact/ Approximate	Type of Heuristic
IDP	Complete solution to reduced problem	Approximate	Dynamic programming
DC	Complete	Approximate	Hill-climbing

C. *Complexity*

DC has a complexity of $O(ng)$. The crossover, mutation, and selection tournaments all have linear costs. IDP is $O(NMi)$. In each iteration, M candidates are evaluated at N grid points for each stage. The number of stages also affects complexity of both algorithms, but this was considered fixed for this project.

The fitness evaluation algorithm, integrate.solve_ivp() from the SciPy library, is the same for both methods. It uses an "Explicit Runge-Kutta method of order 5(4)." [6] The complexity of this method is $O(xy^2)$, where y is the number of variables (in this case 7) and x is the number of steps taken by the solver [7]. The value of x is determined dynamically by the solver, so the complexity depends on the details at each step.

In both cases, fitness evaluation is expensive relative to other steps in the algorithms, but the total cost of this step is still only a linear function of the number of evaluations.

V. EXPERIMENTAL METHODOLOGY

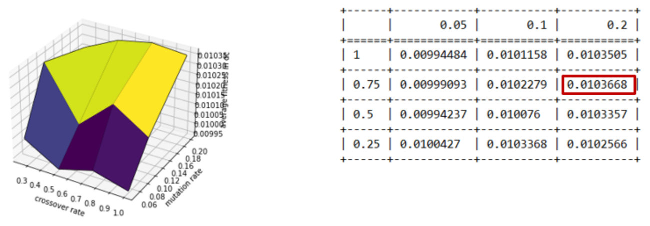
Algorithms were implemented in Python with the NumPy and SciPy libraries. The DC implementation reused code from [8] that was originally derived from [9][10].

A. *DC*

Parameters were set as follows:

- $\sigma = 0.1$
- $\gamma = 0.1$
- $n = 100$
- $g = 50$

To select the best value for mutation rate and crossover rate we ran the algorithm for different values of p_c and p_m. For each combination of assigned values to p_c and p_m, the result (mean values over 10 runs) are displayed in Figure 2. To make the test time reasonable, in this experiment, we set n to 30 and g to 20. In the table, each cell shows the average of best fitness for a given p_c (row) and p_m (column). The red box indicates the cell with the best performance. Based on this experiment we set p_c = 0.75 and p_m = 0.2.

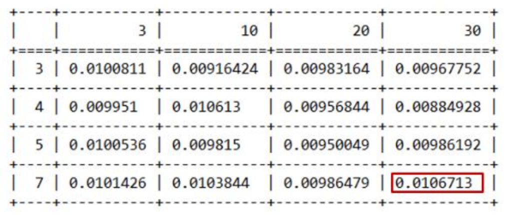


```
+------+------------+-----------+-----------+
|      |       0.05 |       0.1 |       0.2 |
+======+============+===========+===========+
| 1    | 0.00994484 | 0.0101158 | 0.0103505 |
+------+------------+-----------+-----------+
| 0.75 | 0.00999093 | 0.0102279 | 0.0103668 |
+------+------------+-----------+-----------+
| 0.5  | 0.00994237 | 0.010076  | 0.0103357 |
+------+------------+-----------+-----------+
| 0.25 | 0.0100427  | 0.0103368 | 0.0102566 |
+------+------------+-----------+-----------+
```

Fig. 2. Average fitness as a function of mutation rate and cross over rate

B. *IDP*

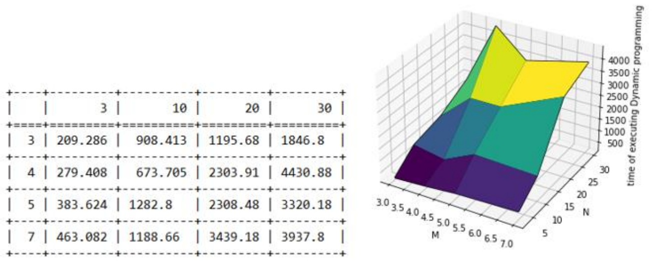
We first had to determine the value of parameters N and M. To select the best value for these parameters we ran the algorithm for four different value of N and four values of M. For each combination of assigned values to N and M, the results are displayed in Figure 3. To make the test time reasonable, in this experiment, we set i to 8. In the table, each cell shows the best fitness for a given M (row) and N (column). The red box indicates the cell with the best performance. Based on this experiment we set M= 7 and N= 30.



```
+---+-----------+------------+------------+------------+
|   |         3 |         10 |         20 |         30 |
+===+===========+============+============+============+
| 3 | 0.0100811 | 0.00916424 | 0.00983164 | 0.00967752 |
+---+-----------+------------+------------+------------+
| 4 | 0.009951  | 0.010613   | 0.00956844 | 0.00884928 |
+---+-----------+------------+------------+------------+
| 5 | 0.0100536 | 0.009815   | 0.00950049 | 0.00986192 |
+---+-----------+------------+------------+------------+
| 7 | 0.0101426 | 0.0103844  | 0.00986479 | 0.0106713  |
+---+-----------+------------+------------+------------+
```

Fig. 3. The fitness value for different assignments to N (column) and M (row)

Figure 4 shows the execution time of IDP is significantly greater with increasing values of N and M. In the table, the first column shows the values of M and first row shows the values of N.



```
+---+---------+---------+---------+---------+
|   |       3 |      10 |      20 |      30 |
+===+=========+=========+=========+=========+
| 3 | 209.286 | 908.413 | 1195.68 | 1846.8  |
+---+---------+---------+---------+---------+
| 4 | 279.408 | 673.705 | 2303.91 | 4430.88 |
+---+---------+---------+---------+---------+
| 5 | 383.624 | 1282.8  | 2308.48 | 3320.18 |
+---+---------+---------+---------+---------+
| 7 | 463.082 | 1188.66 | 3439.18 | 3937.8  |
+---+---------+---------+---------+---------+
```

Fig. 4. Execution time of Dynamic Programming for different values of N (column) and M (row)

Finally to test the IDP method, we used the following parameters:

- $N = 30$
- $M = 7$
- $r = 0.3$
- $\gamma = 0.7$
- Initial u_p = 0.75
- $i = 20$

VI. EXPERIMENTAL ANALYSIS

A. *DC*

The performance of DC when using the best parameters (p_m =0.2 and p_c =0.75) for n =100 and g =50, is shown in Figure 5. F_{min}, F_{avg}, and F_{max} respectively show minimum, average and maximum fitness in each iteration. Final best value of fitness in this experiment was 0.010783 and time elapsed was 4149.02 s.

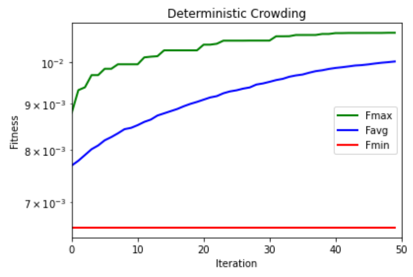


Fig. 5. $F_{\min}$, F_{avg}, and $F_{\max}$ of DC for parameters : Population size = 100, Number of iteration = 50, mutation rate = 0.2 and crossover rate = 0.75

Table III shows four examples of controls found by the algorithm that have a high fitness value. Based on their distance, these points belong to different peaks of the fitness function.

TABLE IV. FOUR SAMPLE INDIVIDUALS WITH A HIGH FITNESS VALUE

Individual	Fitness
[0.67650516, 0.68028349, 0.67512455, 0.9, 0.9, 0.9, 0.89953528 0.9, 0.9, 0.9]	0.01078
[0.66920324,0.69030949,0.67349916, 0.69514756, 0.9, 0.9, 0.89974504, 0.9, 0.9, 0.89949961]	0.01069
[0.68032239, 0.67241294, 0.9, 0.69655815, 0.9, 0.9, 0.9, 0.9, 0.89936344 0.9]	0.01065
[0.89221057, 0.67371608, 0.9, 0.69285349, 0.89949277, 0.9, 0.89843695, 0.9, 0.9, 0.9]	0.01039

B. IDP

The results when using the best parameters (N = 30 and M = 7) for r = 0.3 and i = 20, are shown in Figure 6. $F_{\max}$ shows the fitness in each iteration. Final value of fitness in this experiment was 0.01077282 and elapsed time was 5,177.915.

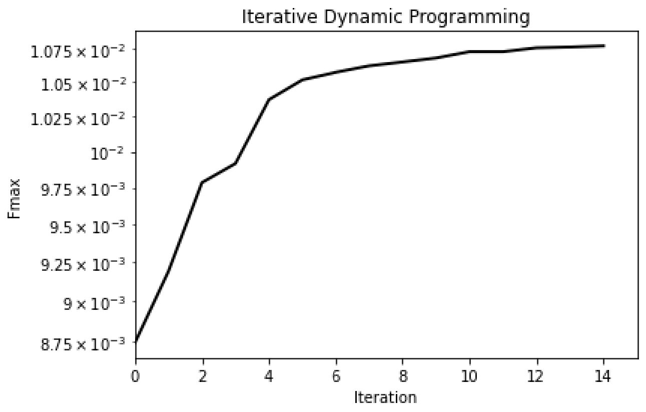


Fig. 6. $F_{\max}$ of DP for parameters : N = 30, M = 7, r = 0.3, *iterations* = 20, γ = 0.7, Initial u_p = 0.75

VII. CONCLUSIONS

Previous studies established the global maximum fitness as 0.0101[1][3]. Our best fitness values were slightly higher. This could be due to differences in the differential equation solver algorithms used. Another possibility is that our method used more significant digits (both in u_i values and in the internal workings of the SciPy solver). [1] reported u_i values to 4 significant digits, compared to the 8 digits we used.

It appears that the best peak found by DC, and the peak returned by IDP, both correspond to the global optimum reported in [1]. The phenotypes are compared in Table IV.

TABLE V. COMPARISON OF BEST MAXIMUM FOUND BY DC WITH GLOBAL MAXIMUM REPORTED IN PREVIOUS STUDY

Phenotype	[1]	DC	IDP
u_1	0.6661	0.67650516	0.66365801
u_2	0.6734	0.68028349	0.68009955
u_3	0.6764	0.67512455	0.68362714
u_4	0.9000	0.90000000	0.89549726
u_5	0.9000	0.90000000	0.89985661
u_6	0.9000	0.90000000	0.89991729
u_7	0.9000	0.899953528	0.89927994
u_8	0.9000	0.90000000	0.89980769
u_9	0.9000	0.90000000	0.90000847
u_{10}	0.9000	0.90000000	0.89992845

A. *Strengths and Limitations*

A single run of DC was able to locate the global maximum. It also returned information on other maxima. The information on other maxima could be useful; for example, if one component of the catalyst blend is more expensive than the other, it could be cost-effective to use a blend other than the one that yields the global maximum.

IDP also located the global maximum. However, the run time required to return this result was slightly greater than that of DC. Reference [1] found IDP to be superior to the NLP method because IDP tends to ignore local maxima that have a fitness that is far less than that of the global maximum. However, it seems to offer less than DC when analyzing a multimodal fitness function. In DC, information on multiple peaks is carried across iterations, whereas IDP only follows one peak across iterations, and this peak may not turn out to be the global maximum.

DC can be improved using a hybrid approach that processes each DC solution with a hillclimbing algorithm to ensure that the individual is at the very top of the nearest peak.

It might be possible to improve the value of IDP by finding a combination of lower M and N that reliably finds the global maximum in less time than DC. IDP can also be run multiple times, possibly using different parameters. This either gives more confidence that the output represents the global maximum, or, if the output varies, it identifies the location of multiple peaks. The use of multiple IDP runs could be made more practical by assigning low values of N and M. Reference [1] reported that even with N = 3, IDP frequently finds the global maximum. Use of multiple IDP runs with low N and M, rather than a single run with high N and M, would give IDP one of the properties that makes DC successful for this problem: multiple solutions are generated to increase the likelihood of finding the global maximum in a complex fitness landscape.

SUPPORTING MATERIAL

All supplemental files are available at https://drive.google.com/drive/folders/1QWQ8mqRuvqogt5guVFPdL0tLTFQytUx3?usp=sharing.

References

[1] Luus, R., Dittrich, J., & Keil, F. J. (1992). Multiplicity of solutions in the optimization of a bifunctional catalyst blend in a tubular reactor. *The Canadian Journal of Chemical Engineering*, *70*(4), 780-785.

[2] Luus, R., & Bojkov, B. (1994). Global optimization of the bifunctional catalyst problem. *The Canadian Journal of Chemical Engineering*, *72*(1), 160-163.

[3] Esposito, W. R., & Floudas, C. A. (2000). Deterministic Global Optimization in Nonlinear Optimal Control Problems. Journal of Global Optimization, 17, 97-126.

[4] Mahfoud, S. W. (1995). A Comparison of Parallel and Sequential Niching Methods. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, 136-143.

[5] O. Nasraoui. *CECS 620: Combinatorial Optimization and Modern Heuristics*, *Complete Evaluation/Solution Based Optimization Methods*, unpublished.

[6] The SciPy Community. (2020). Scipy.integrate.solve_ivp. In *SciPy v1.5.4 Reference Guide*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html.

[7] JayMFleming (https://scicomp.stackexchange.com/users/27138/jaymfleming). (2018). System of ordinary differential equations - time complexity of initial value problem. In *Computational Science Stack Exchange*, https://scicomp.stackexchange.com/q/29373.

[8] Shafaei, S., & LeFevre, J. *CSE 620 Mini-Project 2*, unpublished.

[9] M. K. Heris. Genetic Algorithm in Python – Part A – Practical Genetic Algorithms Series. In YouTube.

[10] M. K. Heris. Genetic Algorithm in Python – Part B – Practical Genetic Algorithms Series. In YouTube.

[11] S. W. Mahfoud. A Comparison of Parallel and Sequential Niching Methods. In Proceedings of the Sixth International Conference on Genetic Algorithms, pages 136-143, 1995.