Systems & Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale,
Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

Introducing Competition in a Multi-Agent System for Hybrid Optimization

Veerawat Udomvorakulchai[a*], Miguel Pineda[a], and Eric S. Fraga[a]

[a] Sargent Centre for Process Systems Engineering, University College London (UCL), Gower Street, London WC1E 7JE, United Kingdom
* Corresponding Author: veerawat.udomvorakulchai.22@ucl.ac.uk.

ABSTRACT

Process systems engineering optimization problems may be challenging. These problems often exhibit nonlinearity, non-convexity, discontinuity, and uncertainty, and often only the values of objective and constraint functions are accessible. Additionally, some problems may be computationally expensive. In such scenarios, black-box optimization methods may be appropriate to tackle such problems. A general-purpose multi-agent framework for optimization has been developed to automate the configuration and use of hybrid optimization, allowing for multiple optimization solvers, including different instances of the same solver. Solvers can share solutions, leading to better outcomes with the same computational effort. Alongside cooperation, competition is introduced by dynamically allocating more computational resource to solvers best suited to the problem. Each solver is assigned a priority that adapts to the evolution of the search. The scheduler is priority-based and uses similar algorithms to those in multi-tasking operating systems. The impact on the number of priority levels is investigated. The framework allows for the use of both metaheuristic and direct search methods. Metaheuristics explore the full search space while direct search methods are good at exploiting solutions. The framework has been implemented in Julia, making full use of multiprocessing. A multiobjective case study on the design of a micro-analytic system is presented. The case study demonstrates the benefits of a multi-solver hybrid optimization approach with both cooperation and competition. The framework adapts to the evolving requirements of the search. Often, a metaheuristic method is allocated more computational resource at the beginning of the search while direct search methods are prioritized later.

Keywords: multi-agent systems, hybrid solution methods, computational resource allocation, multiobjective optimization

INTRODUCTION

Solving optimization problems in process systems engineering is often a challenging endeavor. These problems may exhibit nonlinearity, non-convexity, discontinuity, and uncertainty, which render derivative information unavailable or difficult to obtain. As a result, gradient-based optimization methods may not be appropriate. Instead, black-box optimization methods may be more convenient for tackling such problems as they require only the values of objective functions.

For some optimization problems in process systems engineering there may be another challenge: the computational expense required to evaluate the models may be large, such as when the models are based on differential equations. For problems that have high computational requirements, the number of times the objective function and the constraints may be evaluated will be limited. In these cases, the efficient use of multiple cooperating optimization methods may be necessary. Prior experience suggests that hybrid approaches can lead to better outcomes compared with using a single optimization method, with the same computational effort [1,2]. However, the effectiveness of each method depends on the problem and may actually vary throughout the search, posing difficulties for decision-makers in selecting the appropriate optimization methods. To address this, this paper introduces competition among optimization methods in the multi-agent system to dynamically allocate a finite computational resource. The aim is to have the

most effective methods given access to the resource at the right time during the evolution of the search.

MULTI-AGENT FRAMEWORK

A multi-agent system consists of multiple autonomous computing elements, referred to as *agents*, that interact with one another by sending and receiving *messages* [3]. Such an environment can be used to facilitate the hybridization of optimization methods by assigning optimization methods to agents, with the agents working together to solve the same problem [4]. This concept has been exemplified in works such as [5] and [6].

A general-purpose multi-agent framework for solving optimization problems has recently been developed [1] to automate the configuration and use of hybrid optimization. This framework enables the use of multiple solvers to solve a given optimization problem, whether it is an NLP or MINLP. The solvers may be different optimization methods or variations of the same method with different parameter settings, a concept also used in [7]. Since the parameters associated with an optimization method affect its behavior, and identifying appropriate values often requires significant experimentation, incorporating variations of the same method may be beneficial.

A key distinction of this implementation is the decoupling of solvers from the model evaluation process. In this setup, the solvers do not have direct access to the model, which enables the implementation of a *scheduler* agent that allocates the computational resource to different solvers more efficiently.

The framework is implemented in the Julia programming language, which provides easy access to CPU threads and enables the full use of multiprocessing.

The multi-agent system includes the following agents:

- Multiple instances of a **solver agent** where each instance is a particular optimization method with specific parameter values;
- One or more instances of a **model agent**; this agent evaluates the model when given a point in the search space;
- A **scheduler agent** whose duty is to allocate points requested by the solvers to available model agents, based on the information provided by the analysis agent;
- The **analysis agent** which analyzes all evaluated points and provides the scheduler with information about the search domain, aiding in managing competition among solvers.

One feature of this framework is *cooperation*, where solvers share information at the scheduler agent level with support from the analysis agent. The purpose is to guide the search toward the most promising regions in the search space by distributing useful information about the search domain.

INTRODUCING COMPETITION

The concept of competition is to reward the more effective solvers and penalize those that are not as effective. The reward consists of allowing more evaluations of points in the search requested by the solver agents. It is hoped that this will ensure the best use of the typically limited computational resource available. It is expected that the effectiveness of different solvers will vary during the evolution of the search. Therefore, the reward mechanism needs to be adaptive and based on the actual performance as the search progresses.

Competition among solvers is implemented by the scheduler agent. This agent manages two queues (shown in Figure 1):

- **Availability queue**, a single queue for messages from the model agents indicating their availability to accept a point for model evaluation;
- **Request queue**, consisting of n_l levels of first-in-first-out (FIFO) queues for requests from solver agents for evaluation of a point in the search domain, where n_l represents the highest priority level set by the user.

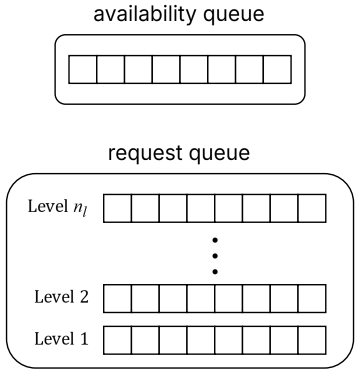


Figure 1. Data structures of the queues managed by the scheduler.

Each solver is assigned a priority [8], ranging from 1 to n_l. The value of n_l is a configuration parameter for the agent system. The impact of the value of this parameter on the performance of the system is discussed below. The priority determines the placement of each request in the request queue and implicitly determines the solver's access to a model agent, with requests at the highest priority level gaining access first. These priorities are dynamic and subject to being adjusted throughout the search based on each solver's performance, also

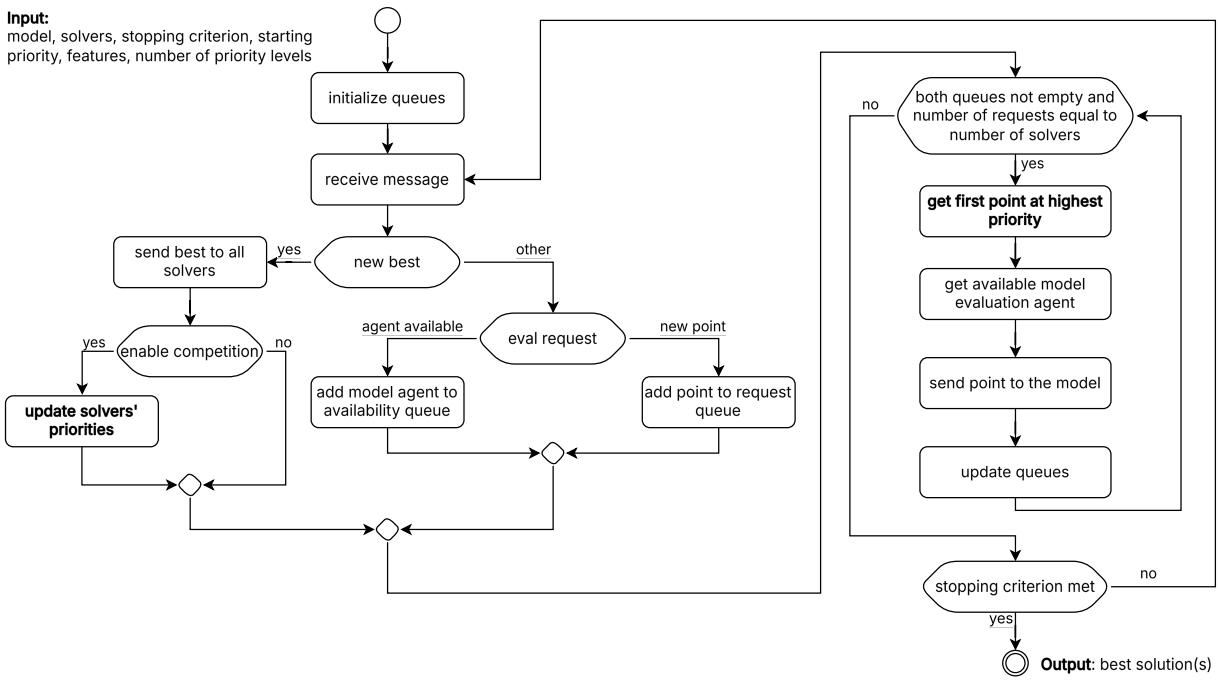


Figure 2: Flowchart illustrating the scheduler agent's process for allocating requests to model agents and managing competition when a new best message is received.

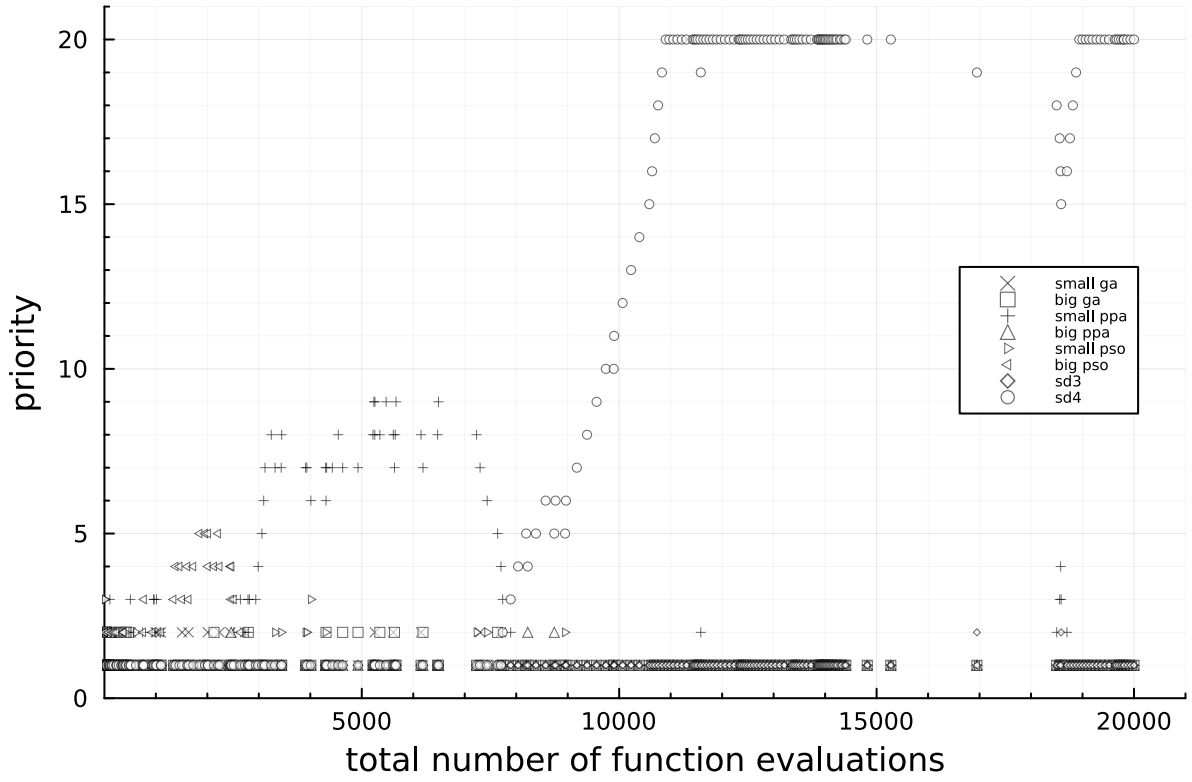


Figure 3: Priority profile of solvers when incorporating both cooperation and competition with 20 priority levels.

described below.

The flowchart in Figure 2 illustrates the main loop of the scheduler in the multi-agent system. Within this loop, the scheduler waits for incoming messages. The element

of competition comes into play when the scheduler receives a new *best* message from the analysis agent, indicating an improvement in the currently best known solution (or the introduction of a new point into the set of non-dominated points). This message contains information about the solution and the solver responsible for it. A solver that is responsible for the improvement is *rewarded* with an increase in its priority, while the priorities of other solvers may be *decreased*, provided they are not already at the lowest level.

Other messages the scheduler may receive can come from a solver agent or a model agent. Based on the type of message, each message is then processed and added to its corresponding queue, either the request queue or the availability queue. For requests from solver agents, this particular message (point) is placed at a specific level of the request queue according to the priority of the solver at the time the request is sent.

The scheduling of the queues occurs when both queues are not empty, and the number of requests is equal to the number of solvers. This assumes that all solvers will continue to submit points for evaluation, which may not be the case for direct search methods in general, as there may be cases where the direct search methods run out of points. For the problems we have considered, the issue has not arisen, but we will, in the future, generalize this condition to ensure that deadlock does not occur. The first point at the highest priority level in the request queue is dispatched to an available model agent. The request queue is then updated: the first entry in each FIFO queue is moved to the end of the next higher priority level if not already at the highest level. This scheduling approach resembles that used in the implementation of multitasking in operating systems [9], ensuring that all solvers are eventually allocated some computational resource.

There are different approaches to adjusting the priority levels and several such approaches have been investigated. One approach involves introducing a decay function for priorities as a function of the number of function evaluations. This decay function, however, introduces an additional configuration parameter. It may be difficult to identify *a priori* an appropriate value of this parameter for any given problem. Another approach considers adjusting the priority increment proportionally to the improvement in the objective function value when a solver finds a new best solution. However, this method is difficult to implement for multi-objective problems as it is difficult to define a metric which quantifies the quality of non-dominated solutions. The approach used in the results presented below is the simplest: reward the solver responsible for the improvement in the best known solution by increasing its priority by one and reducing the priorities of all the other solvers by the same magnitude.

CASE STUDY: A DESIGN OF A MICROFLUIDIC ANALYSIS SYSTEM

An optimization-based design of a micro total analytic system (μTAS) for on-line detection of radionuclides [10] is presented to examine the impact of incorporating the element of competition into the framework. The goal of the design is to detect and identify the presence of two different radionuclides. The identification of designs is a problem of design under *uncertainty*, formulated as a set of differential equations with continuous design variables. This problem involves two objective functions: the first minimizes the system's limit of detection (ϕ), and the second combines the number of false negatives (n_f) and false positives (n_p), with greater emphasis placed on the number of false negatives due to safety reasons.

This problem serves as a good test case for the proposed framework, as the problem is computationally expensive, and the search space is likely multi-modal [10]. Prior experience has demonstrated the benefits of information sharing; however, introducing competition may offer the potential for further improvement in outcomes more quickly.

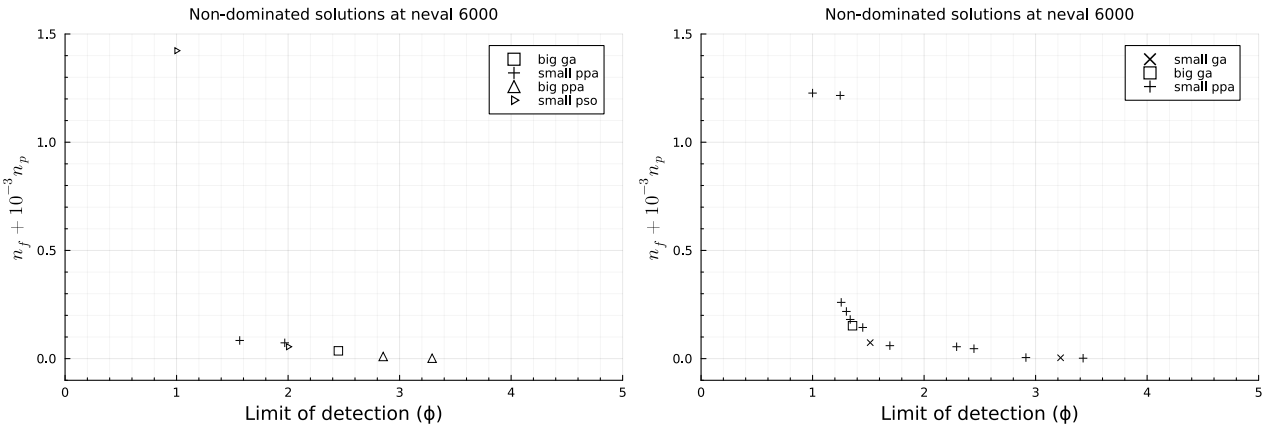
This study employs two types of solvers: nature-inspired metaheuristic methods [11] and direct search methods [12]. The solvers used are as follows:

- Genetic Algorithm (GA)
- Plant Propagation Algorithm (PPA)
- Particle Swarm Optimization (PSO)
- Steepest Descent (SD)
- Coordinate Search (CS)

The key parameter for the GA and the PSO methods is the population size; for the PPA, it is the number of solutions considered for propagation. Multiple instances of the solvers are explored using different parameter values, defined as a function of the base population size, n_s (set to 20 in this case study). For the GA, the population sizes are $2 \times n_s$(small) and $5 \times n_s$ (big). For the PPA, the number of solutions to propagate is $n_s/2$ (small) and $2 \times n_s$ (big). For the PSO, the sizes are n_s (small) and $5\times n_s$ (big).

The instances of direct search methods, on the other hand, are explored by assigning different weighting factors, ω, used in the weighted sum method to aggregate multiple objectives into a single objective function value. Four weighting factors are applied: 0.2, 0.4, 0.6, and 0.8. Direct search solvers treat the base initial population of size n_s as a multi-start, initiating a search from each member of the population sequentially.

None of the implementations here could be considered the *best in class*, but we intend to assess the performance of the collection of algorithms to investigate



(a) Cooperating (b) Cooperating and competing (n_l=20)

Figure 4: Non-dominated solution sets at 6,000 function evaluations with different configuration settings. *Neval* represents the total number of function evaluations, with each solution labeled by the responsible solver. n_f and n_p denote the number of false negatives and false positives, respectively.

the impact of cooperation and competition. The stopping criterion is set to 20,000 total function evaluations by all the solvers.

Impact of competition

To investigate the impact of competition, a set of 10 runs has been conducted incorporating both cooperation and competition, with the number of priority levels, n_l=20. Figure 3 illustrates the priority profiles of each solver during the search, with output printed whenever a new non-dominated solution is identified. Solver instances for the direct search methods are distinguished by the suffixes 1, 2, 3, and 4 corresponding to the weighting factors of 0.2, 0.4, 0.6, and 0.8, respectively.

Two phases of the search can be observed from Figure 3. Initially, the search focuses on exploration, during which metaheuristic solvers, particularly the small PPA, frequently find new non-dominated solutions, as reflected in their increasing priority. This is followed by the exploitation phase, where the SD4 solver (a direct search method) identifies improved solutions, although typically minor improvements. Toward the end of the search, the small PPA identifies some new non-dominated solutions, which are promptly refined by one of the direct search solvers through information sharing, similar to the findings from the previous study [1]. The CS solvers are omitted from the figures as they do not ever find a non-dominated solution.

Sensitivity to the number of priority levels

The number of priority levels, n_l, is a configuration parameter that may influence the performance of the system. Comparing sets of runs with different values of n_l,: 5, 10, and 20, two observations are found. First, a higher number of priority levels results in greater difference between computational resource allocated to the most effective solver and to the others, with the largest difference observed when n_l=20. This occurs because solvers require more iterations to transition from the lowest to the highest priority level. Second, the distinction between the exploration and exploitation phases of the search becomes more apparent with higher n_l values. For example, when n_l=20, the exploration phase lasts shorter compared with n_l=10, whereas for n_l=5, this distinction is blurred. Therefore, it can be concluded that the number of priority levels determines the degree of competition among solvers. A higher number of priority levels closely resembles the individual use of the most effective solver, while lower values result in minimal competition.

Given the motivation of this paper–to make efficient use of a limited resources–we compare sets of non-dominated solutions at 6,000 total function evaluation (Figure 4), where the advantages of incorporating competition are evident. Under computational time constraints, the combined use of both cooperation and competition with 20 priority levels leads to the best outcomes, being able to explore the search space most effectively, with many points covering the objective space.

Although the solution sets in Figure 4 differ, quantifying which is better can be challenging, particularly as the true Pareto front is unknown in this case study. Nevertheless, the results show that incorporating competition alongside cooperation is likely to enhance the exploration process, with almost the same computational effort, and the degree of improvement depends on the number of priority levels. This is particularly beneficial when acceptable solutions are required with limited computational resource. Although the methods are stochastic and repeated attempts will lead to different outcomes, the results presented here are representative.

CONCLUSION

A generic multi-agent framework for hybrid optimization has been developed to automate the configuration and integration of metaheuristic and direct search optimization methods. While information sharing among solvers has been shown to improve outcomes [1], equal allocation of computational resource to all solvers may not be ideal as optimization problems often have evolving requirements at different stages of the search. This motivates the use of competition alongside cooperation in the framework, enabling solvers to be dynamically prioritized based on performance. The framework is able to dynamically allocate computational resource as the search progresses, for instance, prioritizing metaheuristic methods to explore the search space at the start of the search and later shifting emphasis to direct search methods to refine the solutions. The overall outcome is more effective use of limited computational resource. The degree of competition is defined by the total number of priority levels, a key configuration parameter of the system.

Future work will explore more sophisticated scheduling algorithms within the scheduler agent and examine the potential benefits when the scheduler has the knowledge about the properties of the methods and the objective space.

ACKNOWLEDGEMENTS

Mr. Veerawat is grateful for the support provided by the Royal Thai Government Scholarship, which covers his tuition fees and living stipends throughout the duration of his studies at University College London (UCL). The authors also gratefully acknowledge the funding received from UK Research & Innovation through the grant EP/R019223/1 which supported the development of the case study used in this paper.

REFERENCES

1. Fraga ES, Udomvorakulchai V, Papageorgiou LG. A multi-agent system for hybrid optimization. In: 34th European Symposium on Computer Aided Process Engineering / 15th International Symposium on Process Systems Engineering. *Computer Aided Chemical Engineering* 53:3331-3336 (2024). https://doi.org/10.1016/B978-0-443-28824-1.50556-1
2. Fraga ES. Hybrid methods for optimisation. In: Computer Aided Methods in Optimal Design and Operations. Ed: Zilinskas J, Bogle IDL. World Scientific Connect (2006). https://doi.org/10.1142/9789812772954_0001
3. Wooldridge MJ. An Introduction to Multiagent Systems. 2nd Edition. Wiley (2009).
4. Silva MAL, De Souza SR, Souza MJF, et al. Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. *Applied Soft Computing* 71:433-459 (2018).
5. Silva MAL, De Souza SR, Souza MJF, et al. A multi-agent metaheuristic optimization framework with cooperation. In: 2015 Brazilian Conference on Intelligent Systems (BRACIS). 104-109 (2015).
6. Ajmi F, Zgaya H, Othman SB, et al. Generic agent-based optimization framework to solve combinatorial problems. In: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 950-956 (2020).
7. Gebreslassie BH, Diwekar UM. Homogenous multi-agent optimization for process systems engineering problems with a case study of computer aided molecular design. *Chemical Engineering Science* 159:194-206 (2017). https://doi.org/10.1016/j.ces.2016.05.026
8. Peterson JL, Silberschatz A. Operating System Concepts. Addison-Wesley Longman Publishing Co., Inc. (1983). ISBN 0201060973.
9. Madnick SE, Donovan JJ. Operating Systems. McGraw-Hill (1974). ISBN 0070394555.
10. Pineda M, Tsaoulidis D, Filho PIO, et al. Design optimization of microfluidic-based solvent extraction systems for radionuclides detection. *Nuclear Engineering and Design* 383:111432 (2021). https://doi.org/10.1016/j.nucengdes.2021.111432
11. Fraga ES, Nature inspired methods for optimization: A Julia primer for Process Engineering (2022). https://doi.org/10.5281/zenodo.7016482
12. Kelly CT. Iterative Methods for Optimization. SIAM (1999).