University of Manouba
National School of Computer Sciences

Report of
the Design and Development Project

Subject: Cloud Services Composition Assistant

Authors :

Mr. Farouk Bouabid Mr. Hamza Kacem

Supervisors :

Dr. Rim Drira
Mr. Hamdi Gabsi

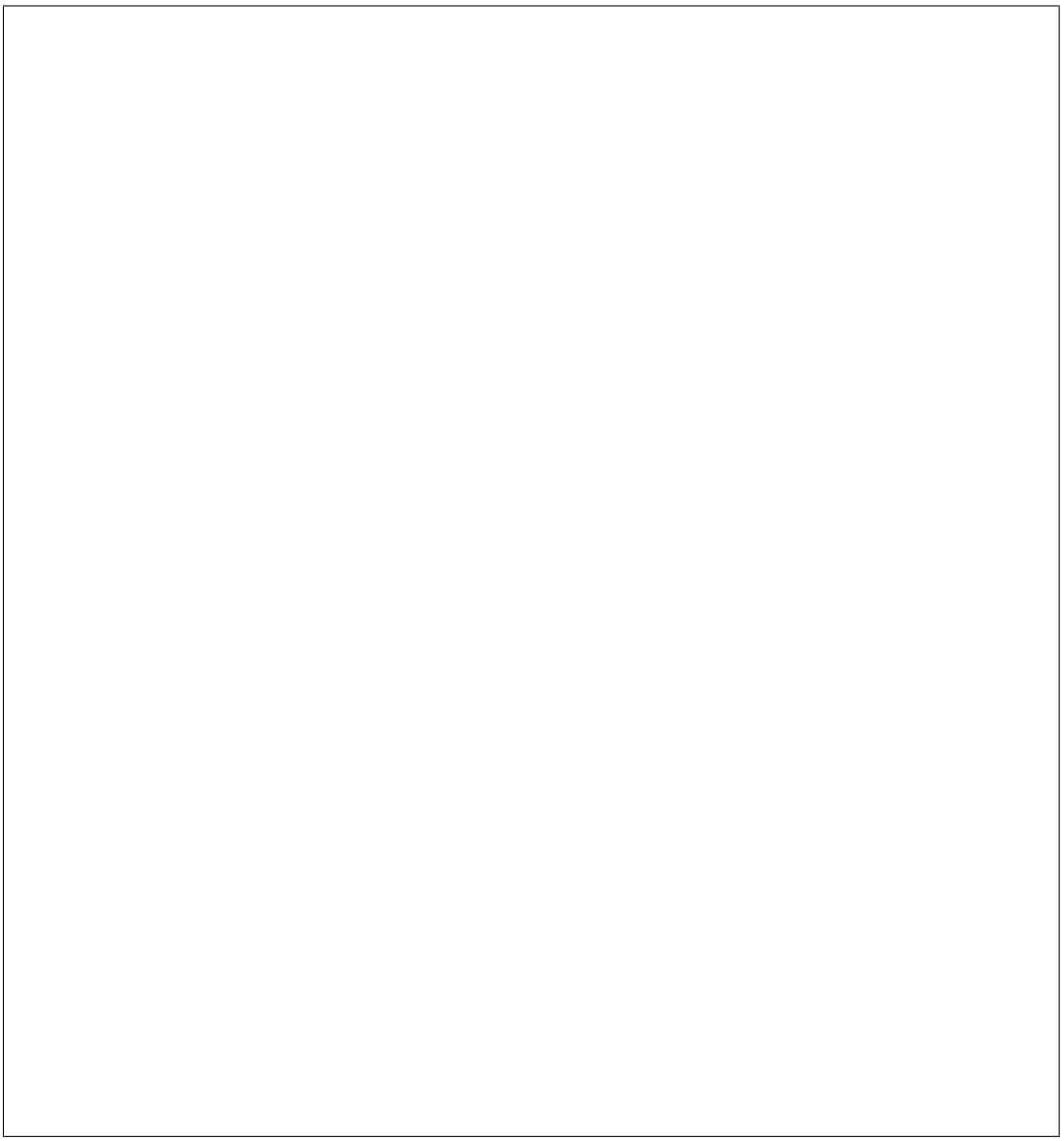
Academic Year : 2019 /2020

Abstract

This report covers the design and the development of a web assistant that handles different optimization-methods to enable a cloud developer to formulate the optimal combination of cloud services that apply to his requirements.
This project is achieved as a second-year design and development project at the National School of Computer Sciences.

Keywords: Cloud services composition, Genetic algorithm, Artificial Bee Colony ...

Approval and signature of the supervisors



Acknowledgement

FIRST and foremost, we would like to express our deepest and sincere gratitude to our supervisors Dr. Rim Drira and Mr. Hamdi Gabsi, for their tremendous efforts, long patience, and full support to us during the entire process of designing this project. It is by dint of their wise guidance, trust, and precious advice that we finalize this teamwork achievement.

Special thanks to all of our teachers in the national school of computer science for their continuous endeavor and dedication to guiding us to the path of success and especially their audacious attempts to save the academic year from the COVID-19 crisis.

Finally, we would like to thank the jury members for the precious time they are dedicating to examine and evaluate our humble project.

Contents

List of Figures

List of Tables

List of abbreviations and acronyms

QoS *Quality of Service*

ABC *Artificial Bee Colony*

GA *Genetic Algorithm*

MCN *Maximum Cycle Number*

SN *Sources Number*

SQ *Source Quit limit*

MOO *Multi Objective Optimization*

MOABC *Multi Objective Artificial Bee Colony*

NSGA *Non dominated sorting Genetic Algorithm*

BSG *Bounded Solution Generator*

ORM *Object Relational Mapping*

GD *Generational Distance*

IGD *Inverted Generational Distance*

HV *HyperVolume*

Pf *Pareto front*

Introduction

CLOUD computing is a process of delivering on-demand IT-capacities (infrastructure, platform or software) to the developers in the form of elastic services without having to control and manage the underlying complexity of the technology.

Nowadays, several cloud services with comparable functionalities are available to developers at different prices and performance levels (together referred to as Quality of Service (QoS) parameters). Developers often find it challenging to select an optimal cloud service that satisfies their requirements. As a single service cannot make them complacent, it becomes necessary to perform a composition of different available services that bridge together to build a composite service satisfying the requirements of the developers.

Thus, in our work, we implement first an optimization-based application that combines a variety of algorithms adapted to the cloud services composition problem, which are the single-objective and the multi-objective optimization algorithms. Second, we develop a user-friendly web interface that enables the developer to access and use these algorithms to obtain service-composition plans that meet her/his requirements.

Our work is presented according to the following outline. In the first chapter, we highlight the most important main concepts related to our work, then we introduce the optimization problem of the search for an optimal composition plan, and the working methodology that we adopt. In the second chapter, we present a theoretical study of the different algorithms that inspire our project.After that, we detail our proposed solution. In the third chapter, we unveil the functional and non-functional requirements and we present the use case and the system sequence diagram. In the fourth chapter, we describe the project design and architecture. In the fifth chapter we go through the achievement of the application. Finally, in conclusion, we outline the main parts of our project as well as the upgrades that can enhance the satisfaction of the customer.

Chapter 1

Preliminary study

In this chapter, we introduce our project, its context, the problem it tackles, and the chosen working methodology.

1.1 Presentation of the project

This project, entitled "Cloud Services Composition Assistant" is elaborated in the context of a second-year study project. It is specified as one of the main subjects in the official program of the National School of Computer Science. Our professors supervise it within four months during the second semester of the second year.

1.2 Main concepts

In this section, we define the most important concepts related to our project.

1.2.1 Business process

According to H.Gabsi and al [3], *"Business Process (BP) development can be defined as the process of constructing a workflow application by composing a set of services performing BP's activities.*
In this respect, Cloud Services (CSs) are being increasingly used in BP development to ensure a high level of performance with a low operating cost."

1.2.2 Cloud services

According to Citrix [19], *"cloud services refers to a wide range of services delivered on-demand to companies and customers over the internet. These services are designed to provide easy, affordable access to applications and resources, without the need for internal*

infrastructure or hardware". Each cloud service has specific qualities of service that are crucial to finding the right tradeoff between services.

1.2.3 Quality of Service (QoS)

According to Danilo Ardagna and al [4], *"QoS denotes the levels of performance, reliability, and availability offered by an application and by the platform or infrastructure that hosts it"*. In our project we are interested in studying four qualities of service :

- The price of using the service per hour.
- The response time that the service takes to achieve its function.
- The availability of the data and service per year.
- The reliability or the probability that a service is operational in a period without any malfunctions.

1.2.4 Service-composition plan

A service-composition plan is a combination of services that interact together to perform a specific business process. Each service performs a defined activity (subtask) at a certain order.

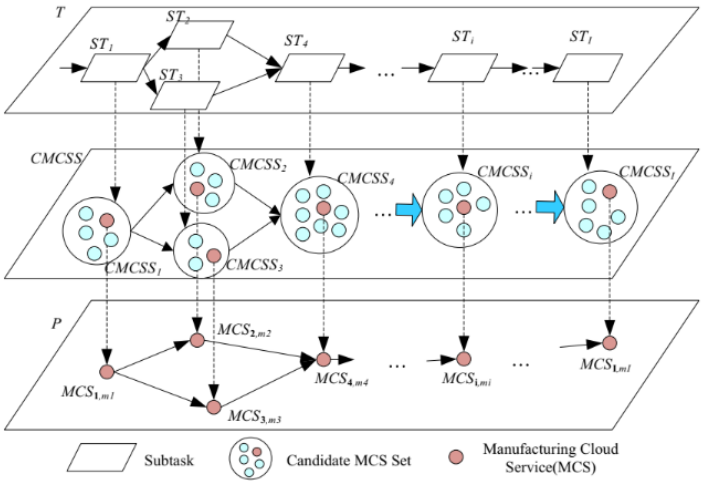


Figure 1.1: Service-composition process [1]

1.2.5 Aggregate QoS

we present a standard way of assessing the aggregate QoS of a service-composition plan according to Wang D and al [5].

Structure	Time	Price	Availability	Reliability
Sequencial	$\sum_{i=1}^{n} q_1(MCS_i)$	$\sum_{i=1}^{n} q_2(MCS_i)$	$\prod_{i=1}^{n} q_3(MCS_i)$	$\prod_{i=1}^{n} q_4(MCS_i)$
Parallel	$\max(q_1(MCS_i))$	$\sum_{i=1}^{n} q_2(MCS_i)$	$\prod_{i=1}^{n} q_3(MCS_i)$	$\prod_{i=1}^{n} q_4(MCS_i)$
Conditional	$\sum_{i=1}^{n} (q_1(MCS_i) * \lambda_i)$	$\sum_{i=1}^{n} q_2((MCS_i) * \lambda_i)$	$\sum_{i=1}^{n} (q_3(MCS_i) * \lambda_i)$	$\sum_{i=1}^{n} (q_4(MCS_i) * \lambda_i)$

Table 1.1: Aggregate QoS formulas

MCS_i is a service in the specific structure, and $\lambda_i = \frac{1}{|MCS|}$ where $|MCS|$ is the number of services in the structure.
Using these calculation methods, we can evaluate and compare composition plans, and thus execute optimization algorithms.

1.2.6 Metaheuristic optimization

A metaheuristic is a general description of an algorithm dedicated to solving difficult (typically NP-hard problem) optimization problems for which there are no classical solving methods.
Researchers have been using algorithms that are inspired by biological evolution such as the Genetic algorithms (GA), or from nature like using the intelligence behavior of a group (swarm) of animals such as the Artificial bee colony algorithm (ABC). These methods achieved considerably good results to these complex problems.

1.3 Problem setting

Our objective is to assist the cloud application developer in finding an optimal solution for a business process, which is a cloud-service-composition plan that optimizes the four main QoS described in (1.2.3), and the matching state, which is the level of conformity in both input and output (type and number) between the service and the corresponding activity. A service with a precise matching state is two times more eligible to be selected than a service with an over matching state.
In addition to that, certain constraints that are introduced by the developer must not be violated, such as the budget limit or the time limit of the entire service-composition plan.

As we wade through the service composition process, we realize that processing all possible

solutions to find the optimal one would take exponential time. For instance, if there are m abstract services and n candidate services for each abstract service, the number of possible service composition plans would be n^m. Since finding the optimal solution regarding a given set of constraints is an NP-Hard problem, we need to find an amicable solution, i.e., a near-optimal solution that satisfy constraints, which turned out to be an optimization problem.

Therefore finding the optimal service-composition plan involves meta-heuristic algorithms to be applied, by which we try to minimize the price and response time, and maximize the availability and reliability of the service-composition plan.
Moreover the execution time of these algorithms must be minimal (low scalability), and the solution found must be sufficient for the needs of the developer (High optimality).

In addition to that, developers may favor the optimizing of a particular QoS over another or may prefer a wide variety of solutions where each composition plan optimizes a different QoS. As a response to these requirements two different types of optimization must be considered: The single-objective algorithm that uses the preferences of the developer to maximize a weighted sum of all the QoS and delivers a single optimal solution. And the multi-objective algorithms that require no preferences from the customer and proceed to give a larger number of solutions, where each optimizes a particular QoS more than others.

1.4 Working Methodology

In this segment, we introduce the agile methodology as well as the scrum methodology that we have chosen for organizing and assigning the work that needs to be achieved each week.

1.4.1 Agile Methodology

Agile methods refer to a group of software development techniques based on iterative development, where requirements and solutions evolve. Agile methods generally promote a disciplined project management process that favors continual examination and adaptation, a leadership philosophy that encourages cooperation, a set of practices dedicated to rapidly providing high-quality software, and a business strategy that aligns development with customer demands and company objectives.

Agile methods are described as a kind of software development process based on four core elements outlined in the Agile Manifesto as:

✓Individuals and interactions over processes and tools.

✓Working software over comprehensive documentation.
✓Customer trust regarding the negotiation of contracts.
✓Responding to change according to schedule.

The most widely-used and suitable Agile methodology for our project is the scrum methodology.

1.4.2 Scrum

Scrum is an iterative and incremental framework for managing product development by dividing it into user stories that represent desired features (functional requirements). These user stories are performed in time-bound iterations called sprints.
Scrum describes a flexible, holistic product development strategy where a development team works as a unit to reach a common goal, challenges the traditional, sequential approach to product development, and encourages teams to coordinate themselves by facilitating physical co-location or near online collaboration as well as frequent face-to-face contact between all team members.

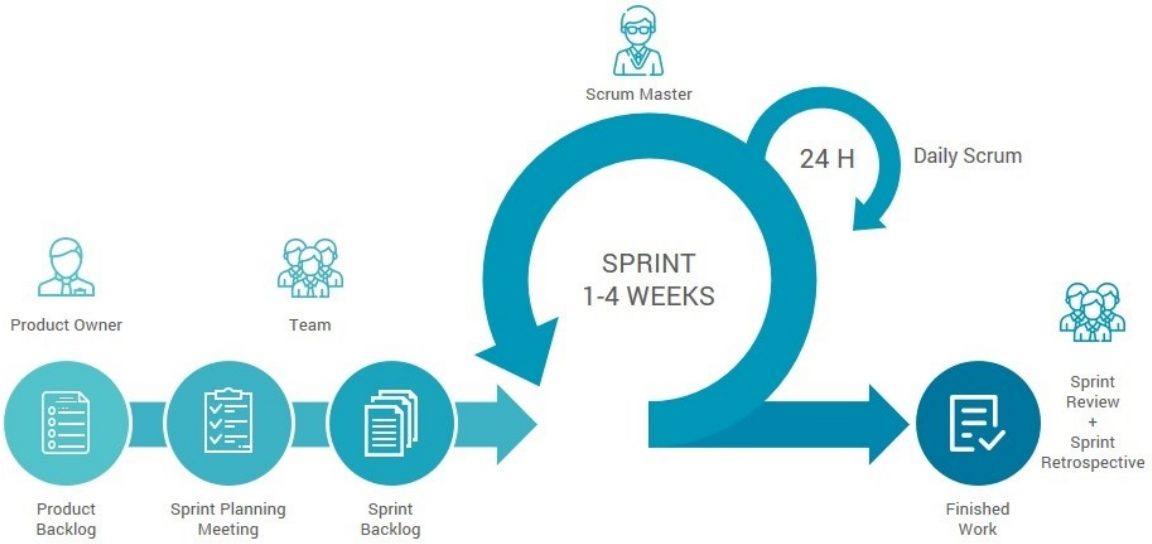


Figure 1.2: Workflow of scrum methodology [14]

1.4.3 Organisation

The SCRUM methodology involves 3 main roles which are:

- Product owner: in the majority of projects, the product manager (Product owner) is the head of the client project team. He is the one who denies and prioritizes the user stories and selects the duration and sprint backlog of each sprint based on the values (loads) he receives from the rest of the team.
- Scrum Master: A true facilitator on the project, he ensures that everyone can work to the best of their abilities by removing obstacles and protecting the team from external disruptions. He also pays particular attention to respecting the different phases of the SCRUM.
- Team: the team organizes itself and remains unchanged for the duration of a sprint. it must do everything possible to deliver the product.

1.4.4 Chosen Methodology

We decide to use the Scrum strategies to match the amount of work we have and our team capability as this strategy helps us to :

- Improve the quality of the deliverables
- Cope better with change (and expect the changes)
- Be more in control of the project schedule and state
- Release unfinished but usable product

Conclusion

This chapter includes a presentation of our project, an overview of the problem setting. In addition to that, we present the agile methodologies and we choose the most convenient approach for our project development.

Chapter 2

Theoretical study

In this chapter, we start by presenting the studied algorithms, their advantages, and limits. Therefore, we specify our proposed solution and used technique.

2.1 Single objective algorithms

The purpose of a single-objective optimization problem is to find the best solution for a specific one or many QoS criteria.
According to João M.P.Cardoso and al [6] *"we can further combine multiple criteria into a single-objective optimization problem by defining the single-objective cost function as a weighted sum of the normalized costs associated with each of the QoS criteria"*. given as:

$$Cost = \sum_{i=1}^{k} w_i \times normalized(Cost_i) \tag{2.1}$$

where w_i is the weight associated with i^{th} QoS criteria and $\sum_{i=1}^{k} w_i = 1$
This function gives a fitness score to each solution (fitness function) which enables the identification of optimality and the comparison between solutions.
Two of the widely-used single-objective algorithms are the ABC and GA algorithms

2.1.1 Artificial bee colony

According to scholarpedia [20] *"The ABC introduced by Karaboga [7], is one of the swarm intelligence algorithms used to solve optimization problems which is inspired by the foraging behaviour of the honey bees."*
The bee colony consists of three types of bees, the employed bees and the onlooker bees, whose role is to exploit rich food sources neighboring the hive, and the scout bees who randomly explore resources in other positions.

This algorithm is based on four main steps, and can be represented the following pseudo-code :

1: Initialization phase
2: Repeat
3: Employed bees phase
4: Onlooker bees phase
5: Scout bees phase
6: Memorize the best solution achieved so far (best fitness)
7: until(Cycle = MCN)

Initialization phase

In this step initializing food sources $\overrightarrow{x_m}$, (m=1...SN), each $\overrightarrow{x_m}$ vector is composed of n variables $(x_{mi}, i = 1...n)$ which are to be optimized.

$$x_{mi} = l_i + \text{rand}(0, 1) * (u_i - l_i) \tag{2.2}$$

where l_i and u_i are the lower and upper bound of the parameter x_{mi} , respectively.

Employed bees phase

Employed bees fly to the current food sources in memory, and then try to improve the nectar quantity by searching for better neighboring resources $\overrightarrow{v_m}$. After evaluating the new solution, if $fit(\overrightarrow{v_m}) > fit(\overrightarrow{x_m})$, then the previous food source $\overrightarrow{x_m}$ is replaced by $\overrightarrow{v_m}$. Each solution has a limit of attempts to improve. After draining a food source to its limit, the scout bees start operating to replace it with a new one.
The food source can be exploited using the following equation :

$$v_{mi} = x_{mi} + \phi_{mi}(x_{mi} - x_{ki}) \tag{2.3}$$

where $\overrightarrow{x_k}$ is a randomly selected food source, i is a randomly chosen parameter index and ϕ_{mi} is a random number within the range [-1,1].

Onlooker bees phase

After all employed bees achieve the search process, the sharing information begins, where the food sources and their position information is shared with the unemployed bees. Onlooker bees choose a food source depending on the probability values measured using the

fitness values provided by employed bees. The probability value p_m with which $\overrightarrow{x_m}$ is chosen by an onlooker bee can be calculated by using this equation :

$$p_m = \frac{fit_m(\overrightarrow{x_m})}{\sum_{m=1}^{SN} fit_m(\overrightarrow{x_m})} \tag{2.4}$$

where $fit_m(\overrightarrow{x_m})$ is the fitness value of the solution $\overrightarrow{x_m}$.

After a food source $\overrightarrow{x_m}$ for an onlooker bee is probabilistically chosen, a neighbourhood source $\overrightarrow{v_m}$ is generated using equation (2.3), and its fitness value is computed. As in the employed bees phase, a greedy selection is applied between $\overrightarrow{v_m}$ and $\overrightarrow{x_m}$

Scout bees phase

Scout bees is an important component to control the exploration process [8]. After the limit is achieved, the employed bee is converted to a scout to search for new food sources. For instance, if solution $\overrightarrow{x_m}$ has been abandoned, the new solution discovered by the scout who was the employed bee of $\overrightarrow{x_m}$ can be defined by (2.2).

2.1.2 Genetic Algorithm

Genetic algorithms commonly produce good results for optimization and search problems by imitating biological operators such as mutation, crossover, and selection. John Holland [9] introduced genetic algorithms based on the concept of Darwin's theory of evolution.

In a genetic algorithm, a population of possible solutions (individuals) to an optimization problem is evolving toward better solutions. Each individual has a set of properties (chromosome). Traditionally, solutions are represented in binary, but other encodings are also possible in different contexts.
This algorithm is based on modifying the initial generation of individuals by producing new offsprings that maintain better fitness values. The following pseudo-code is a simplified representation of this process :

1: Initialization phase (population + evaluation)
2: Repeat
3: Selection phase
4: Reproduction Phase
5: Mutation phase
6: Evaluating offsprings and building new generation
7: until(Cycle = Maximum generations number)

Initialization phase

The process begins by generating a random set of individuals, which is called a Population. Each individual is a solution to the problem you want to solve, represented as a string of 0s and 1s. Thus Each individual is assigned a fitness score.

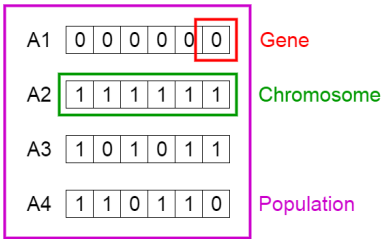


Figure 2.1: Population, Chromosomes and Genes [15]

Selection phase

The idea of the selection phase is to select the fittest and most qualified individuals (parents), and build the next generation based on their genes by performing biological operators (mutation and crossover).

Reproduction phase

Crossover is the most vital phase in a genetic algorithm. It can be performed via various methods: uniform crossover, single-point crossover, or two-point crossover.
In the uniform crossover for example, each gene (bit) is selected randomly or by a probability ratio from one of the corresponding genes of the parents' chromosomes.

Mutation phase

After the reproduction, new offsprings can be subject to a mutation with a low probability. This signifies that some of the bits in their chromosomes can be flipped.
Mutation occurs to preserve diversity within the population and inhibit early convergence.

Building new generation phase

Generating the new offsprings increases the size of the population. Thus, to preserve a fixed number of qualified individuals within the population, A selection based on fitness scores is performed, and the new generation is built.
The algorithm delivers optimal solutions if the population has converged (does not generate offsprings which have better fitness scores than the individuals in the previous generation).

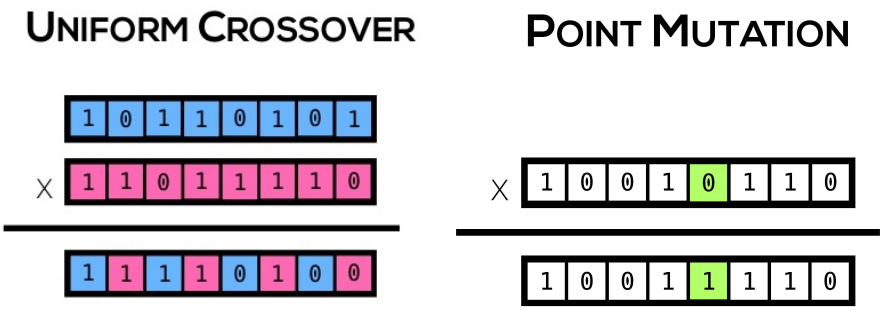


Figure 2.2: Genetic operations [16]

2.2 Multi objective algorithms

Multi-objective optimization (MOO) has emerged as a preferable approach to tackle problems involving more than one QoS criteria to be optimized concurrently. MOO algorithms generally produce a set of non-dominated solutions or (optimal Pareto), representing optimal trade-offs between given criteria.
According to Emrah and al [10] *"The objective function $F(x)$ is defined as a vector of n multiple conflicting functions"* :

$$F(x) = (f_1(x), f_2(x), ..., f_n(x)) \tag{2.5}$$

Dominating solution

According to Emrah and al [10] *"For two solutions y and z, y dominates z if and only if, y is not worse than z in all objectives and better than z in at least one objective."*

Pareto front

According to Kalyanmoy Deb and al [2] *"Pareto front is a collection of non-dominated solutions where one objective cannot be improved without detriment to another."* objective.[2]

Non-dominated sort

According to Kalyanmoy Deb and al [2] *"The Non-Dominated Sort aims to classify individuals according to their level of non-domination."* Solutions that are not dominated by any other solution form the first Pareto front. To find the solutions in the next front, the solutions appearing in the first front are temporarily eliminated, and the same procedure is repeated.

```
fast-non-dominated-sort(P)
for each p ∈ P
  S_p = ∅
  n_p = 0
  for each q ∈ P
    if (p ≺ q) then                    If p dominates q
      S_p = S_p ∪ {q}                  Add q to the set of solutions dominated by p
    else if (q ≺ p) then
      n_p = n_p + 1                    Increment the domination counter of p
  if n_p = 0 then                      p belongs to the first front
    p_rank = 1
    F_1 = F_1 ∪ {p}
i = 1                                  Initialize the front counter
while F_i ≠ ∅
  Q = ∅                                Used to store the members of the next front
  for each p ∈ F_i
    for each q ∈ S_p
      n_q = n_q − 1
      if n_q = 0 then                  q belongs to the next front
        q_rank = i + 1
        Q = Q ∪ {q}
  i = i + 1
  F_i = Q
```

Figure 2.3: Fast non dominated sort [2]

After applying non dominated sort on solutions, many different approaches deal with how to select surviving individuals for the next generation. One of these approaches is the crowding distances.

Crowding distances

Crowding distances are introduced whenever there is a need to select some solutions from a given front. They give a measure of closeness in performance to other solutions. The crowding distance in one dimension for any individual is the distance between the next lowest and next highest score in that dimension.

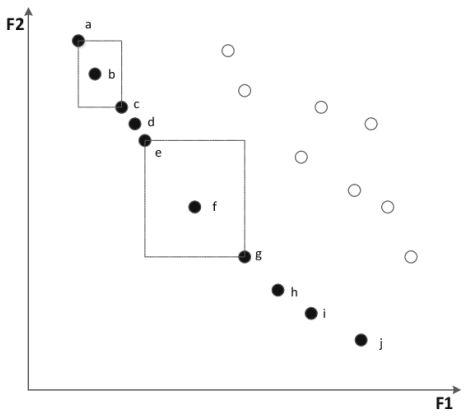


Figure 2.4: Crowding distance [17]

Each individual will have a crowding score for each dimension (objective), which will eventually be summed to give the final crowding score.

2.2.1 MOABC

The MOABC algorithm follows the same process as the single objective ABC featuring the three main phases. However, the MOABC algorithm evaluates the fitness of the resources differently.

$$fit(\overrightarrow{x_m}) = \frac{dom(\overrightarrow{x_m})}{SN} \tag{2.6}$$

where $dom(\overrightarrow{x_m})$ is the number of solutions dominated by $\overrightarrow{x_m}$

The most eligible food sources are the less dominated and more dominating. Therefore the non-dominated sorting is applied after each phase of the algorithm to ensure retaining the fittest and most promising solutions as given in this pseudo-code :

```
Initialization phase
Repeat
    # Employed bees phase
    Generate a new neighboring food source with (2.3) and add it to the solutions
    Apply non dominated sorting to solutions
    Retain solutions based on their rank and crowding scores
    # Onlooker bees phase
    Probabilistically select food sources (2.4)
    Generate a new neighboring food source with (2.3) and add it to the solutions
     Apply non dominated sorting to solutions
    Retain solutions based on their rank and crowding scores
    # Scout bees phase
    if limit reached
        Generate a new random food source with (2.2) and add it to the solutions
        Apply non dominated sorting to solutions
        Retain solutions based on their rank and crowding scores
until(Cycle = MCN)
```

2.2.2 NSGA-II

According to Kalyanmoy Deb and al [2] *"NSGA-II is one of the most popular multi objective optimization algorithms with three special characteristics, fast non-dominated sorting*

approach, fast crowded distance estimation procedure and simple crowded comparison operator."
According to Yusliza Yusoff and al [11] *"Unlike the single objective optimization GA, NSGA-II simultaneously optimizes each objective without being dominated by any other solution."*

Binary solution generator

According to Emrah and al [10] *"For each solution, the BSG creates four neighbor solutions using sequentially the Two-point crossover and Two-way mutation."*
1. Two-point crossover: Two positions are randomly determined on binary parents x_i and x_k. Everything between the positions of xi is copied to x_k to generate the first offspring. Then, everything between the positions of x_k is copied to x_i to generate the latter one. In this way, two offsprings are generated.
2. Two-way mutation: First, a number within the range of 0 and 1 is randomly generated. If the generated number is greater than 0.5, a position with value 1 is chosen and its value is set to 0. Otherwise, a position with value 0 is chosen and its value is set to 1.

Commonly, NSGA-II can be described as the following steps :

1: Initialization phase (population + evaluation)
2: Applying non-dominated sort and calculating crowding scores
3: Repeat
4: Selection phase based on rank and crowding scores
5: Reproduction Phase (BSG)
6: Applying non-dominated sort and calculating crowding scores
7: Building new generation based on rank and crowding scores
8: until(Cycle = Maximum generations number)

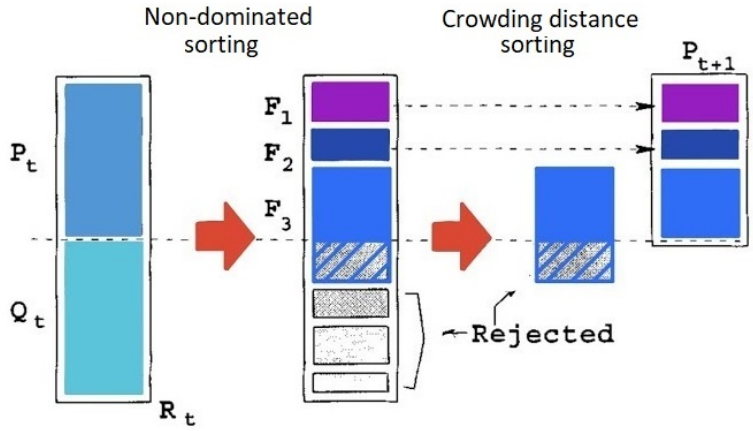


Figure 2.5: Building new generation [18]

2.3 Proposed solution

In the cloud services composing problem, the desired solution(s) is a structure made of candidate services (composition plan) that performs a specific business process. The search space will be the set of possible composition plans that can be constructed based on the available candidate services.

2.3.1 Strength and weakness of studied algorithms

According to Gerhardt and al [12] *"ABC based algorithms have the advantages of having a simple concept, easy implementation, high flexibility, and an efficient exploration."* Nevertheless, the accuracy of the optimal result which cannot meet the requirements sometimes, and the basic resource generation method present a genuine need to add other features to the basic theory of these algorithms. Moreover, in our cloud services composing problem, a discrete representation of the search space is an absolute necessity. Thus, we need to consider a fresh perspective to accommodate the search operation.

Genetic based algorithms have also many advantages, including flexibility, parallel search capacity (each population can explore the search space in many directions), and very efficient biological operators that enable the manipulation and exploitation of discrete complex solutions. However, According to Xin-She Yang and al [13] *"genetic algorithms are easily affected by any inappropriate choice of parameters that will make it difficult for the algorithm to converge."* The premature convergence and the lack of exploration can also be a significant disadvantage when dealing with larger search spaces.

2.3.2 Hybridization

Hybridization is merging desired features of each algorithm, so that the resulting aggregate algorithm is more efficient and more adequate to the context.
Since both the ABC and GA implementations are inefficient alone to tackle the problem of optimal composition plan search, we propose a hybrid approach that inherits specific operations or phases from each algorithm in both the single-objective and multi-objective optimization domain. This approach is elaborated in the achievement chapter.

Conclusion

This chapter is intended for the theoretical study of the algorithms that we use during our project. In the next chapter, we specify the different features granted by our web application.

Chapter 3

Requirements Analysis and Specification

In this chapter we aim to analyze and present our project formally. We start by introducing the actors alongside both the functional and non-functional requirements of the project. We exhibit the main scenarios by presenting the use case of the application and the sequence diagram of the main use case scenario.

3.1 Requirements Analysis

3.1.1 Actors

An Actor is an external element that interacts with the system, making decisions and initiatives. In our application, we distinguish two main actors :

- The cloud developer : uses the application to generate composition plans suitable for his/her needs
- The administrator : Sets the parameters of the optimization algorithms

3.1.2 Functional Requirements (Product backlog)

Functional requirements describe the functionalities or services of the system. They depend on the type of software, the intended users and the type of system where the software will be used.
All the required functionalities are described in user stories to which we give an estimation (user story points) of the work than needs to be achieved.

All the user stories are summed in a product backlog which is presented in the following table :

ID	Story	Acceptance Criteria	Estimation (Fibonacci)
1	As a developer I want to access the home page So I can learn more about the application	1- About us section 2- Why choose us section 3- FaQs section	5
2	As a developer I want to create an account So I can store my operations data	1- Register button 2- Username input 3- Email input 4- Password input 5- Email confirmation	2
3	As a developer I want to login So I can review my operations history	1- Login button 2- Email input 3- Password input	1
4	As a developer I want to review my operation history So I can download my operations data	1- Operations list 2- Download input file 3- Download output file	2
5	As a developer I want to logout So I can hide my data from other users	1- Logout button	2
6	As an administrator I want to set the parameters of the algorithms So i can tune the performance of the algorithms	1- Access parameters table button 2- Modify and save preferences	1
7	As a developer I want to choose an optimization method So I can specify my QoS preferences	1- No QoS preferences method selection 2- QoS preferences method selection 3- Description of each method	3
8	As a developer I want to import JSON file So I can specify the business process	1- Upload file from device 2- Submit button 3- Input file description	3
9	As a developer I want to execute the single-objective algorithm So I can optimize my QoS preferences	1- Single objective algorithm	21
10	As a developer I want to display the result So I can verify and download it	1- Show service-composition plan 2- Show the Aggregate QoS 3- Download button	2
11	As a developer I want to execute the multi-objective algorithm So I can get a variety of solutions	1- Multi-objective algorithm	13

Table 3.1: Backlog

3.1.3 Non functional requirements

Non-functional requirements represent the constraints that the system must meet in order to guarantee its proper functioning and, eventually, its validation with the customer. Among these needs, we can list :

- **Extensibility:** There may be a possibility to add new features or modify existing ones.

- **User-friendliness:** The application must be user-friendly, simple and easy to handle even by non-experts, a developer must be able to use all the functionalities of the application in a time not exceeding 10 min.

- **Maintainability:** The application code must be understandable and well commented.

- **Efficiency:** The system must optimize the use of memory and computing resources (CPU).

- **Reliability and Viability:** The application must be functional regardless of any circumstances surrounding the user.

3.2 Requirements Specification

3.2.1 Use case diagram

Within this section, we explain the use case diagram, which reveals the potential users of the application and demonstrates their different ways of interacting with it.

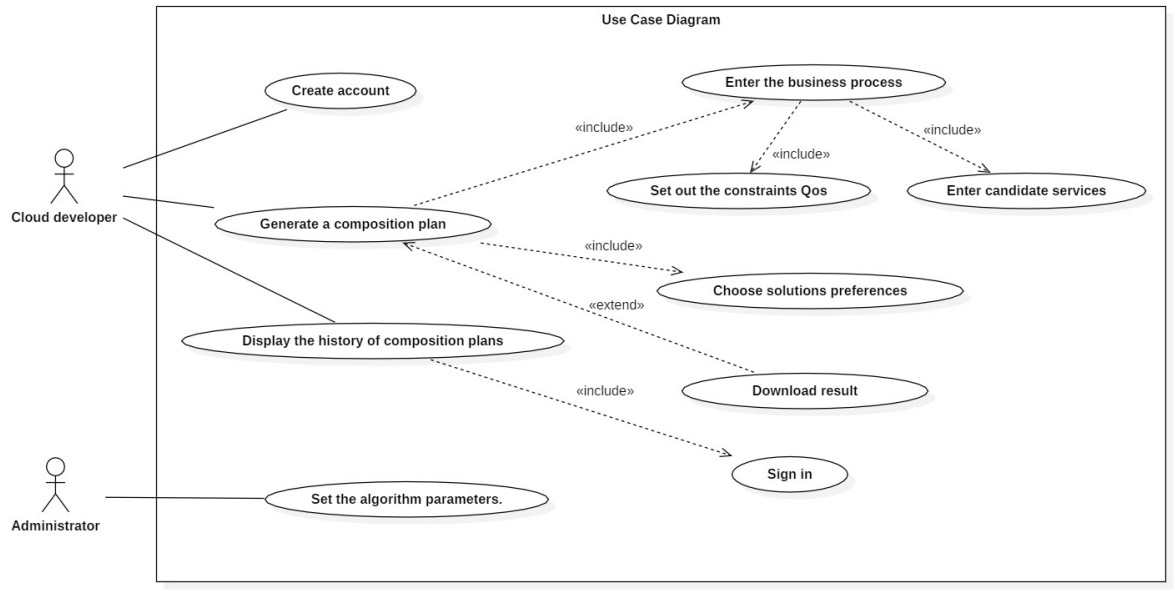


Figure 3.1: Use case diagram

3.2.2 "Generate a composition plan" sequence diagram

In this section, we present a simple sequence diagram showing, from a temporary point of view, the interaction between the developer and the various components in our application to enable the generate composition plan functionality.

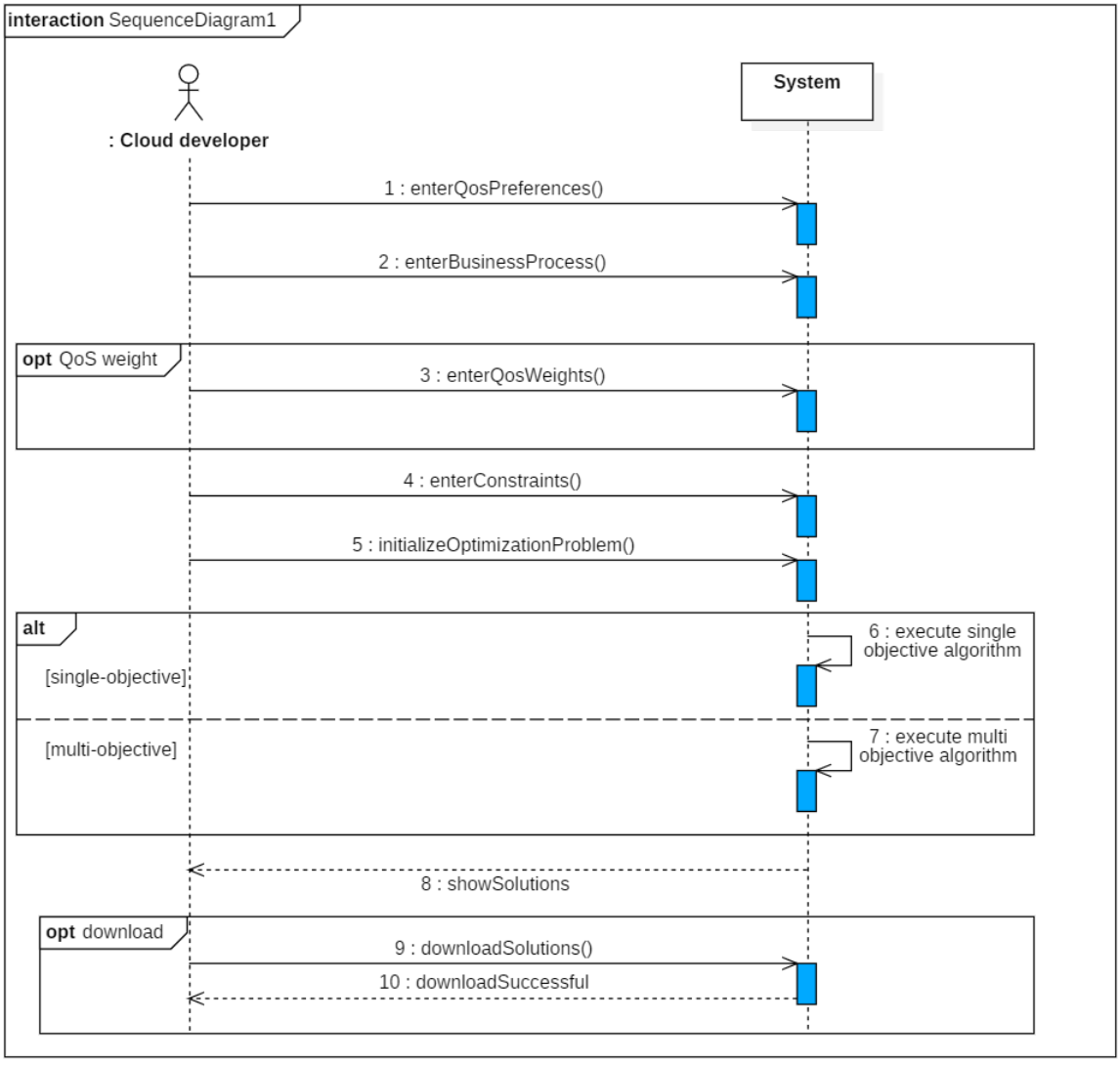


Figure 3.2: Sequence diagram of use case: generate a composition plan

The developer has the option of registering. In this case, he will become able to store the data (Business process) he submitted before and the corresponding application results.

Therefore he will possess the ability to retrieve stored data and reuse it. As for the process of generating a solution, the developer has to submit his preferences regarding the type of solutions he prefers and the most important QoS to optimize. Besides, he has to submit a file that designates the business process he wishes to achieve and its candidate services. Hence the application initializes the optimization problem and executes the suitable algorithm. As a result, the user is now able to download the solutions, which are automatically stored in the database.

3.3 Sprints planning

In this section, we present the planning of each sprint. Our project is mainly achieved in three sprints (beside sprint 0 for design) described below.

Story ID	Sprint 1	Sprint 2	Sprint 3	Estimation
1	X			5
2	X			2
3	X			1
4	X			2
5	X			2
6	X			1
7	X			3
8	X			3
Total				19
9		X		21
10		X		2
Total				23
11			X	13
Total				13

Table 3.2: Sprints planning

Conclusion

During this chapter, we present the actors, specify the requirements of our web-application and we reveal the use cases and scenarios of our project. In the next chapter, we unveil the process of conceiving the application by presenting the design of the project.

Chapter 4

Design

In this chapter, we focus on the main architecture and the details of its components to achieve the appropriate result described in the requirements analysis and specification.

4.1 Global design

4.1.1 Physical architecture

In this section we describe the physical architecture of our application in this section, which is the architecture of 3-tiers.

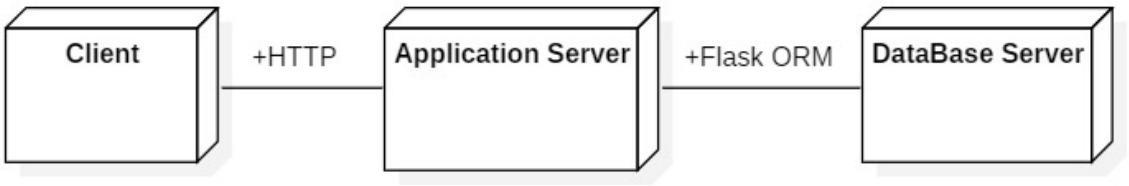


Figure 4.1: 3-Tier architecture

What characterizes this physical architecture is the development and maintenance of the three layers of presentation, treatment, and data on separate devices as independent modules. These three layers are, respectively, the client tier, the Web server tier, and the database server tier.
In this context, we should be mindful that the client tier will only submit queries and obtain replies from the site server since it is not permitted to access the database server explicitly because The middle-ware HTTP protocol assures this connection between the client and the Web.On the other hand, the level of the web-server is only allowed to send

requests and receive answers from the database server since the Flask object relational mapping (ORM) assures this communication.
What made us opt for this choice was the ability of this architecture to efficiently and safely provide multiple access to the application. It also makes the system run faster by loading up the pages.

4.1.2 Logical architecture

The Flask micro-frame meets the model-view-template (MVT) concept pattern, which is a design pattern closely build on the model-view-controller basics.
Flask takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with handling the template, which is an HTML file mixed with Flask Template Language (Jinja2).

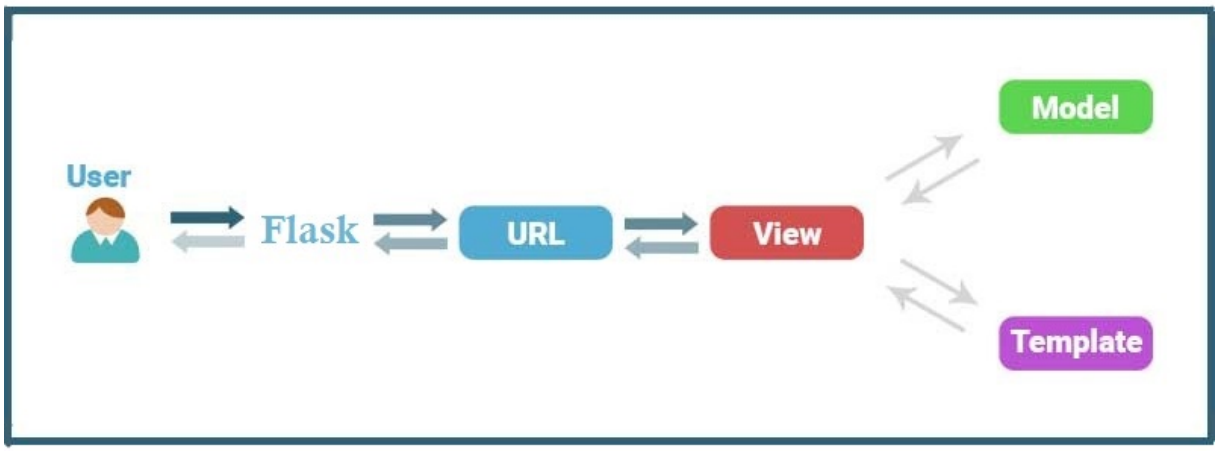


Figure 4.2: MVT Architectural Pattern

- **Model:** It is the access layer for the data, featuring input forms, data access, validation, and relationships.
- **View:** It is the layer of the control logic. By containing the logic that accesses the model and manages the appropriate template, this layer is responsible for the coordination.
- **template:** It is the layer of presentation, responsible for how something should appear on a web page..

In our application, the model contains the data related to users. The templates contains files used for building the front-end of our application while the view is responsible for providing the appropriate response to every request related to the application and playing the coordinator role between the template and the model.

4.2 Detailed design

In this section, we explore the design of our application in greater depth. In the first part, we present the package diagram to show the arrangement and organization of the web-application. Whereas, in the second part, we present the class diagram that describes the static view of our algorithms.

4.2.1 Package Diagram

This diagram shows the packages that compose the web application and the dependencies between these packages.

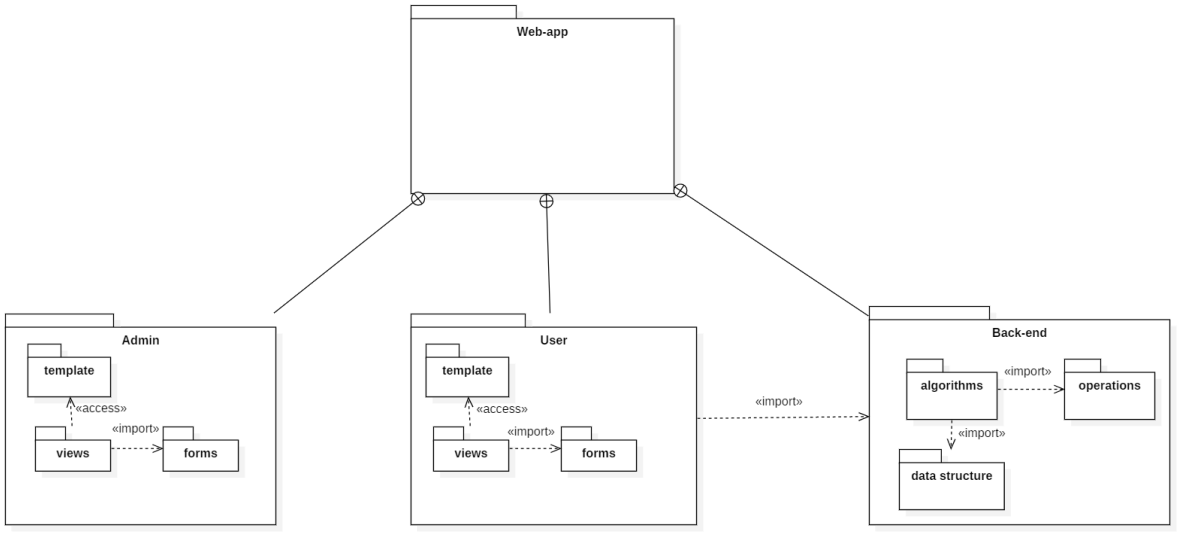


Figure 4.3: Package diagram

This package diagram shows that our application has two main components, the admin and user packages that present the handling of the client-server interaction, and the algorithms handling package that initializes the problem, creates the different objects, imports the necessary modules and executes the algorithms.
The purpose of the sub-packages templates and views are previously explained in the logical architecture and designate the general structure of the flask framework. Meanwhile, the sub-packages operations and data structure have an essential role in the algorithms:

- **data structure:** defines all the classes used by the algorithms to generate and manipulate solutions (CompositionPlan objects).
- **operations:** contains all the genetic manipulations, the fitness evaluation, and the different methods used by the algorithms such as the non-dominated sorting and the crowding distance evaluation.

4.2.2 Class diagram

This diagram shows the classes that compose the data structure package and the dependencies between these classes.

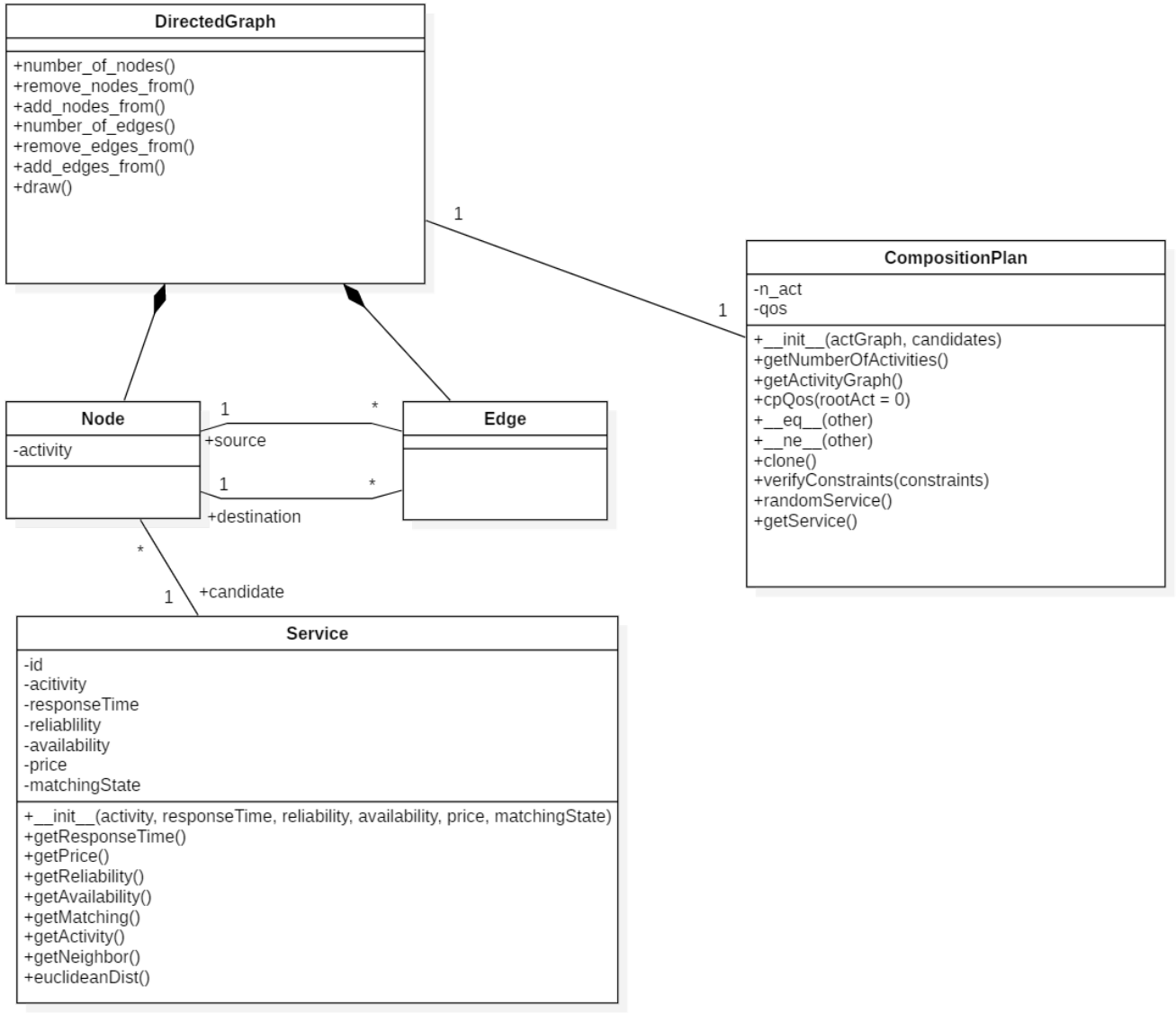


Figure 4.4: Class diagram of data structure Package

A DirectedGraph, which represents an essential attribute of a composition plan object, is composed of nodes and edges. Each node can be a source as well as a destination to many edges. Furthermore, a node has an attribute called activity that represents the abstract activity to perform and an attribute that assigns the chosen candidate service for this activity. Meanwhile, the edges hold a specific value (-1, 0, 1) that indicates the type of

relation between the source node and the destination node: 0 for a sequential type relation, 1 for a parallel type relation and -1 for a conditional type relation.

The next diagram describes the chronological sequence of events that lead to the generation of a desired solution.

4.2.3 Sequence diagram

This sequence diagram gives a detailed account of the communication between the different web pages and the Flask server.

- **Web pages:** The client-end where we display all information and features.
- **Views:** Manages all information, receives requests from the Website, provides responses, renders templates and connects to the database.
- **Generate:** A module responsible for initializing and solving the cloud composition problem.

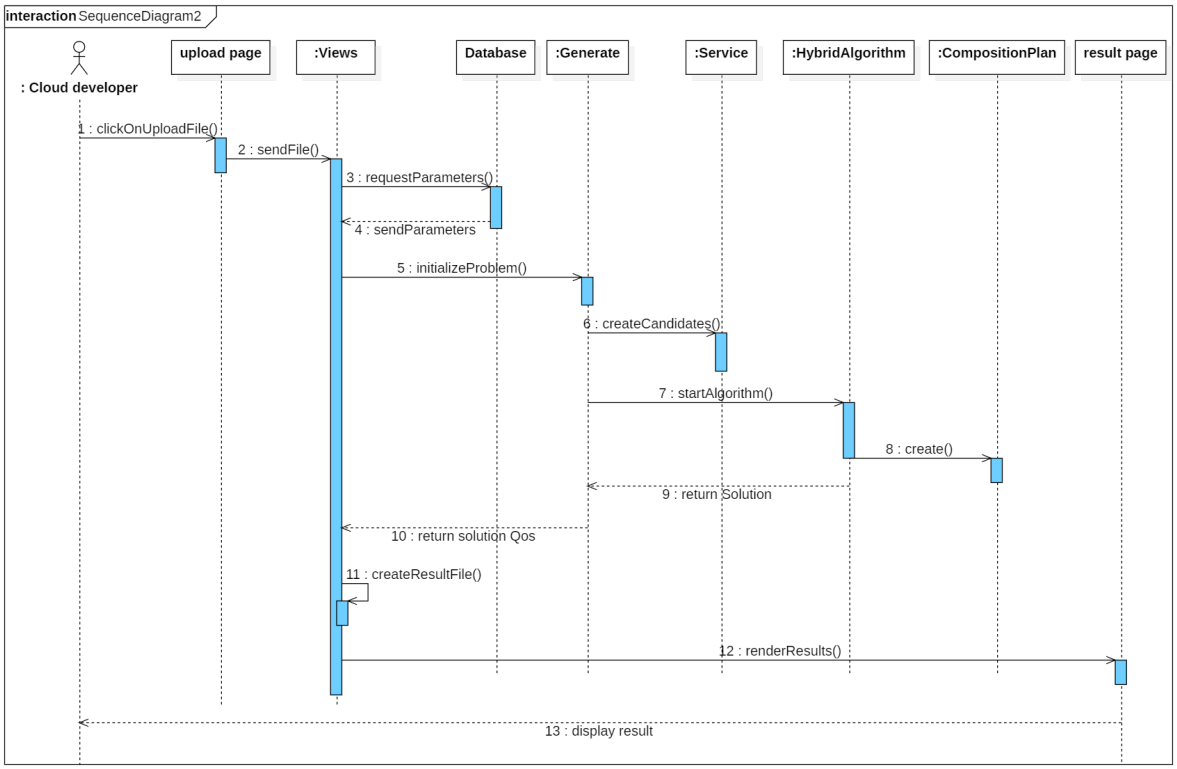


Figure 4.5: Solution generating sequence diagram

Conclusion

During this chapter, we demonstrate a detailed design-approach of our application highlighting the interactions between the packages and the primal components of the algorithms.

Chapter 5

Achievement

This chapter analyzes each sprint achieved during our work, demonstrating the implementation of each component of our application.

5.1 Sprint 1 : Platform

The sprint starts with creating the home page of the application, the login and register pages, the admin dashboard, the method choosing interface, and finishes with the history page of the previous operations of each account.

5.1.1 Home page

The home page includes different sections that provide both the information and guidance that a developer needs to use the application.

- About us section: A section that introduces the application and its purpose.
- Why choose us section: A section that presents the advantages that a developer gains of using our application.
- FAQs section: A section that answers commonly asked questions about the functionalities included in our application.

The home page also includes a direct link to services web page where the developer can choose to login and/or use the application.

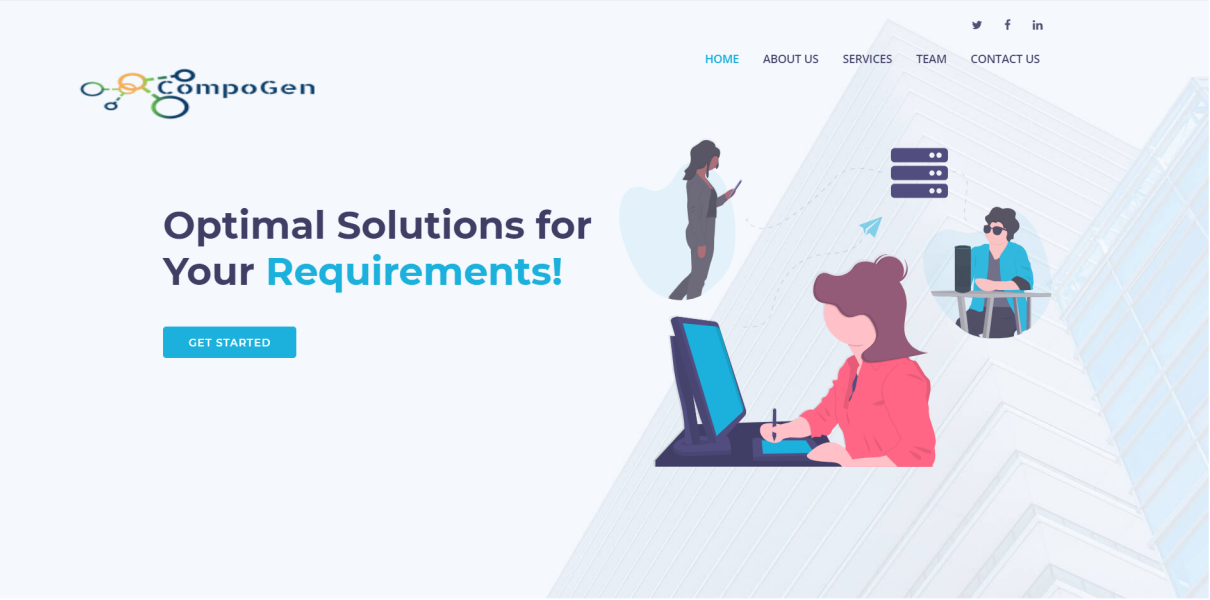


Figure 5.1: Homepage section

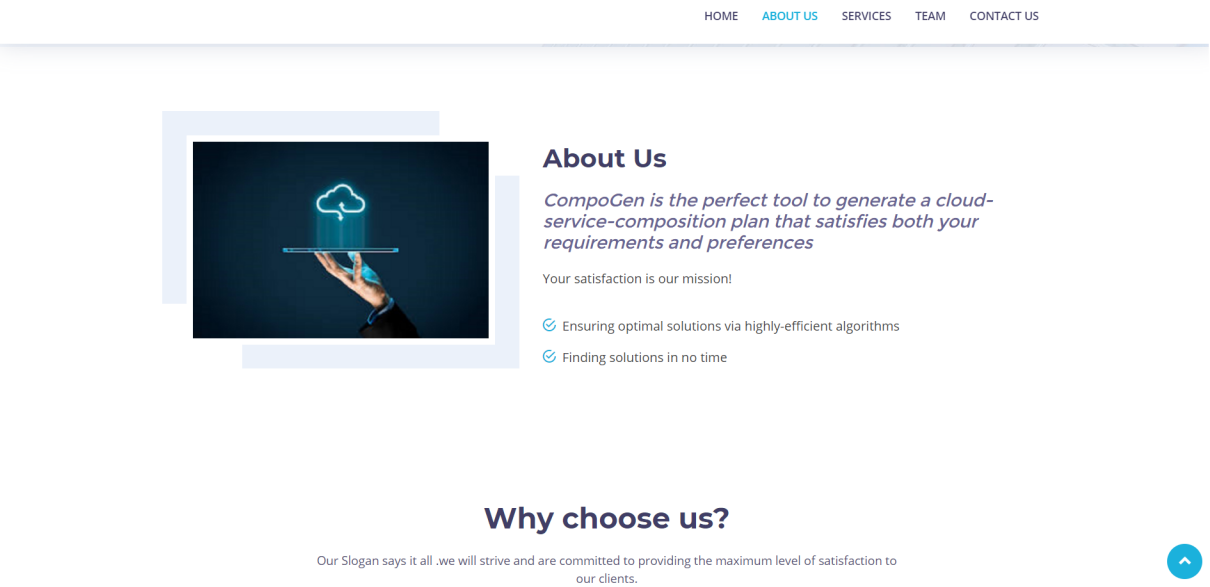


Figure 5.2: About us section

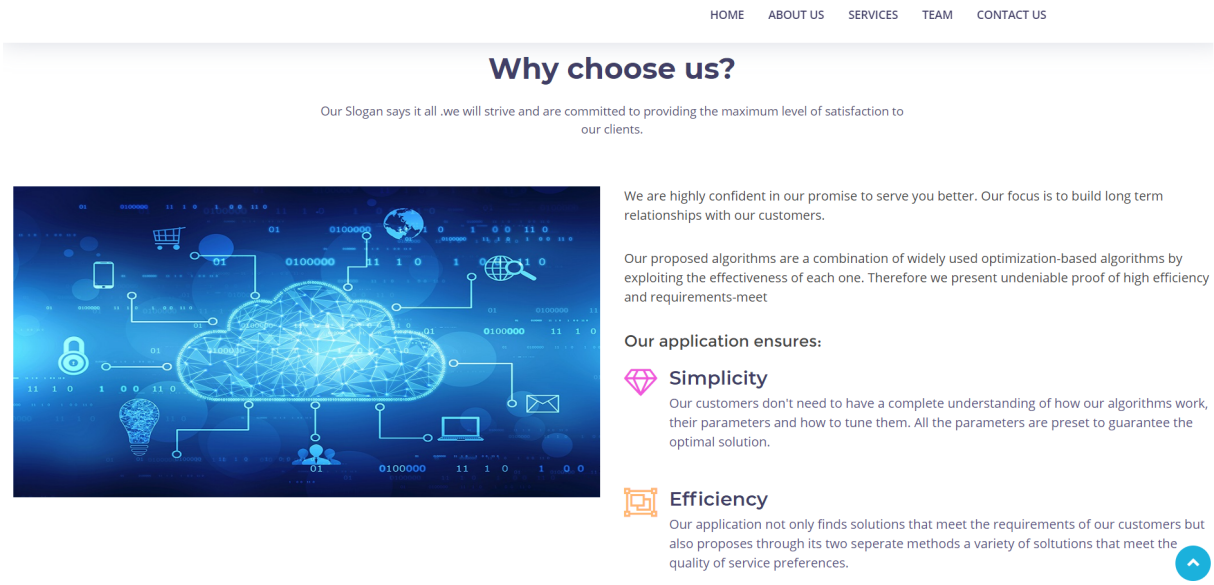


Figure 5.3: Why choose us section

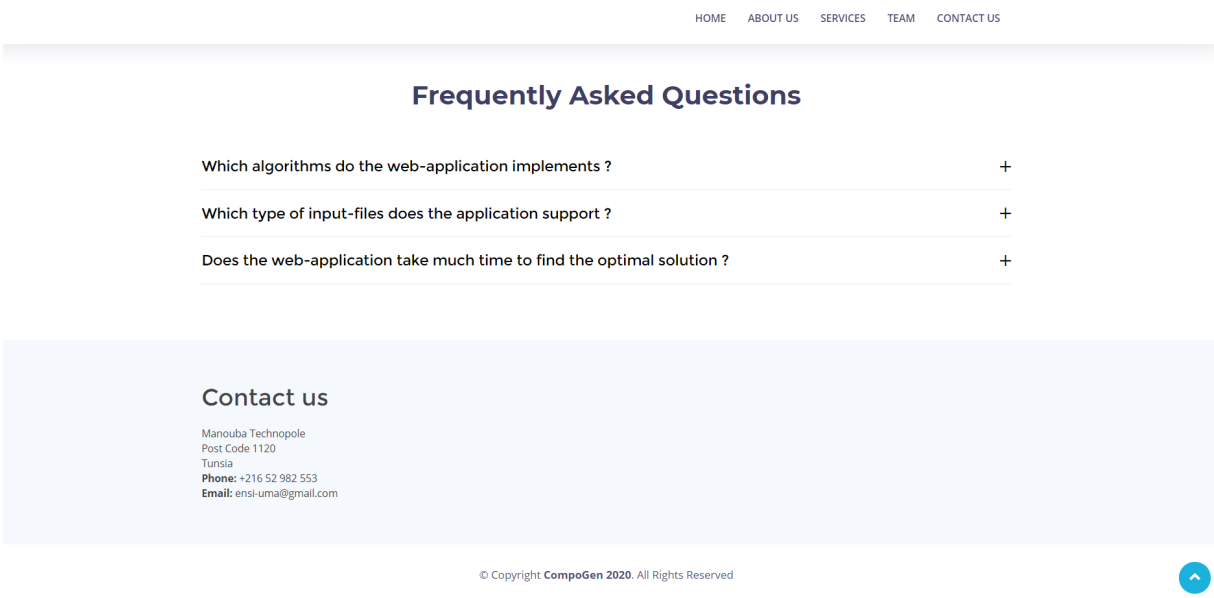


Figure 5.4: FaQs section

5.1.2 Register/Login page

This interface allows the developer to create an account which grants him the storage possibility of the input and output files of previous operations.

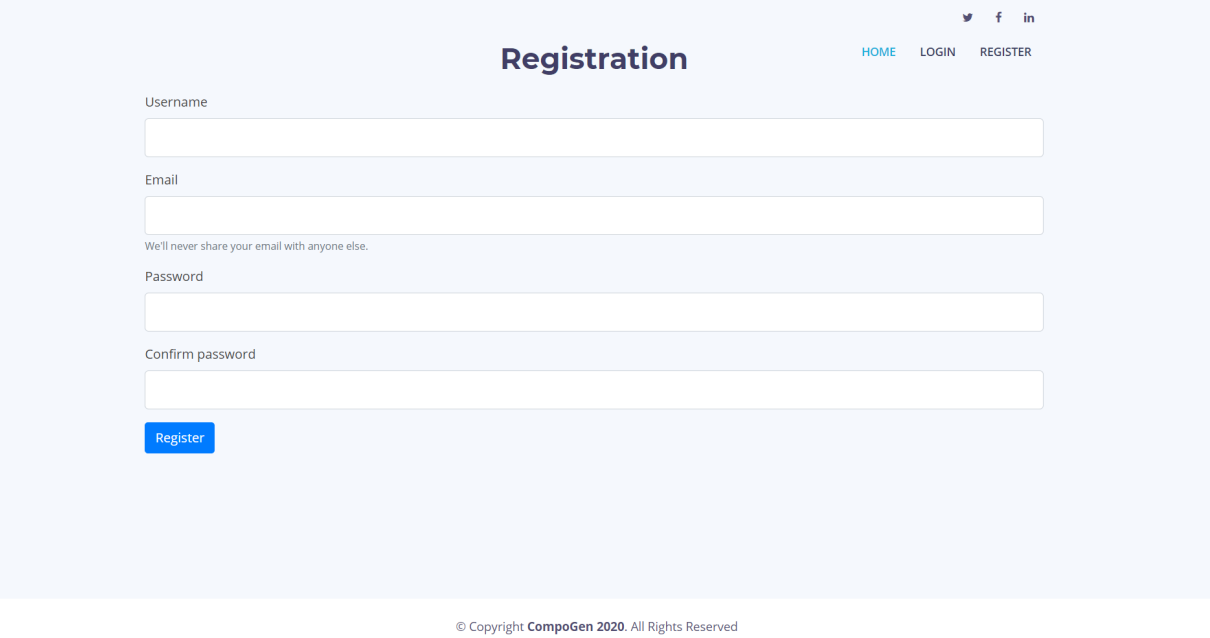


Figure 5.5: Register section

After a successful registration, the user can log in to access the previously mentioned feature

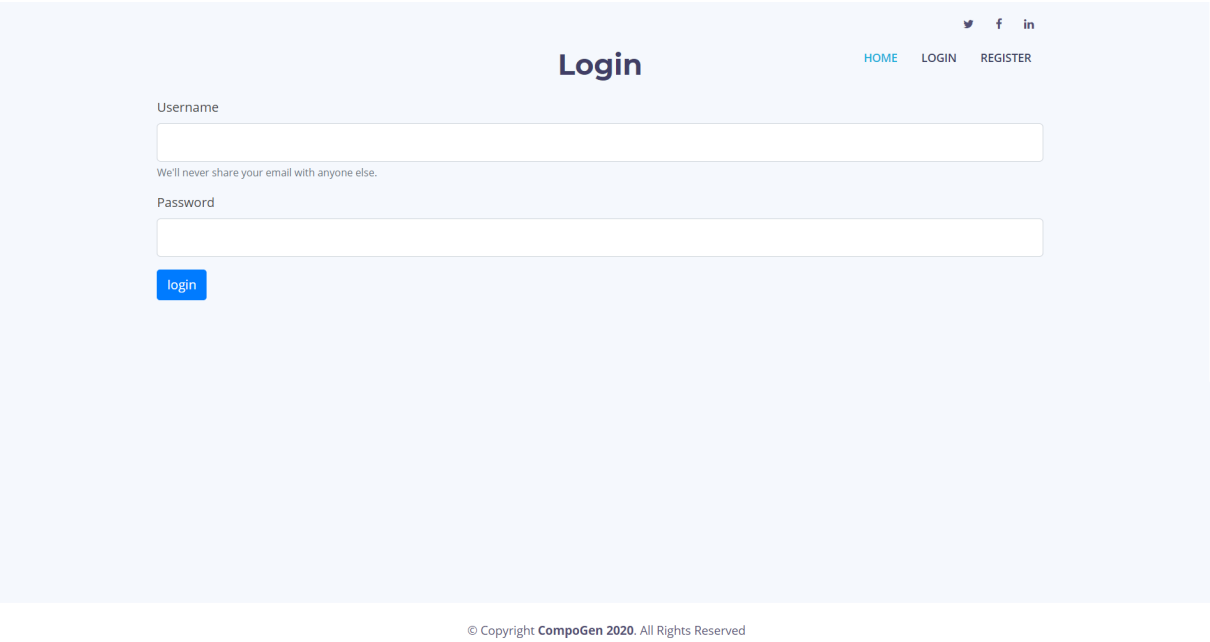


Figure 5.6: Login section

5.1.3 History page

The registered developer can review his operations history and download the input or the output file of any specific business process he requested before.

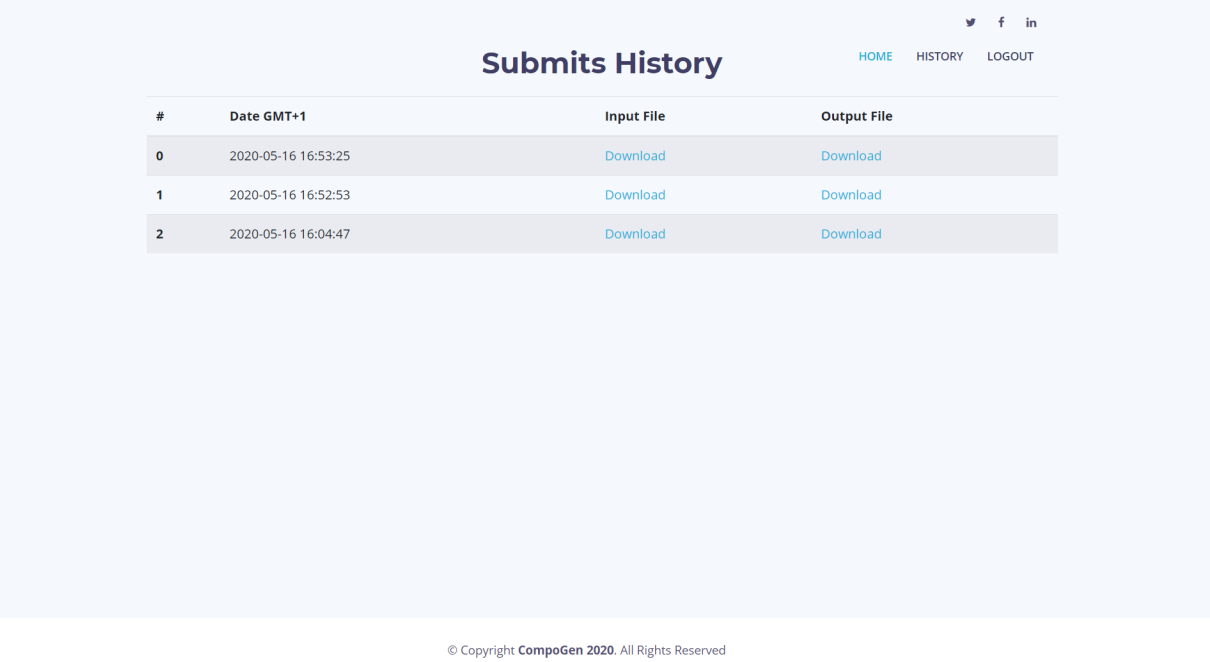


Figure 5.7: History page

5.1.4 Method choosing page

Once he/she clicks on the "get started" or "services" button, the developer is transferred to the method choosing page. An interface that requests that the developer specifies whether he has QoS preferences (single-objective algorithm proceeds in this case) or not (multi-objective algorithm proceeds in this case).

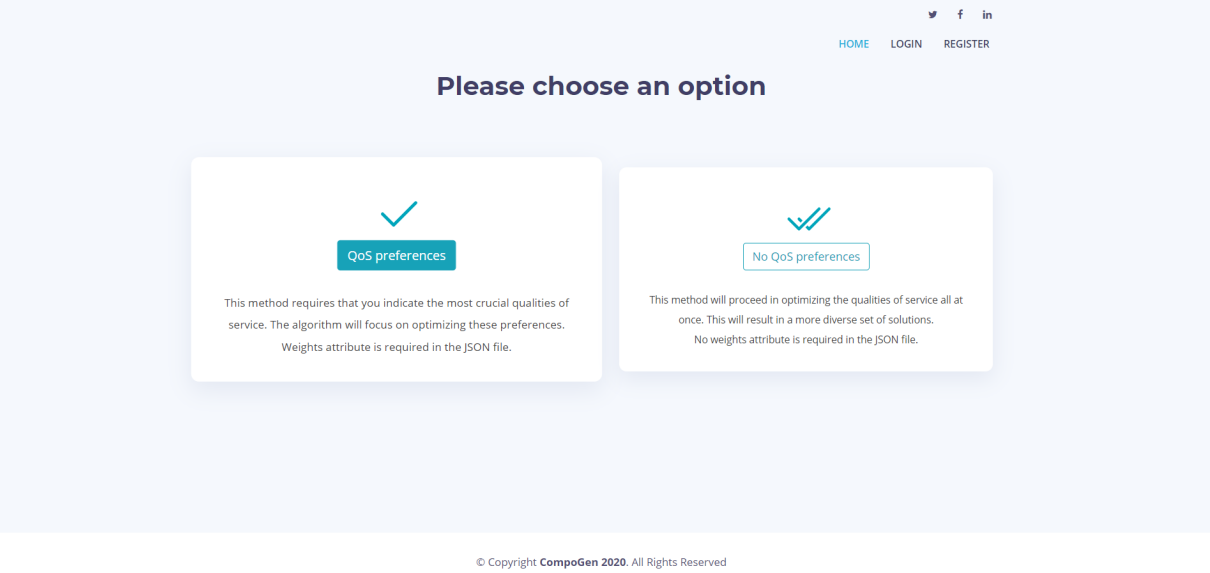


Figure 5.8: Choosing method page

5.1.5 Business process submit page

After choosing a method, the application requires that the developer submits a JSON file that includes: the desired business process, the candidate services, the constraints, and the weights(for the single-objective algorithm).

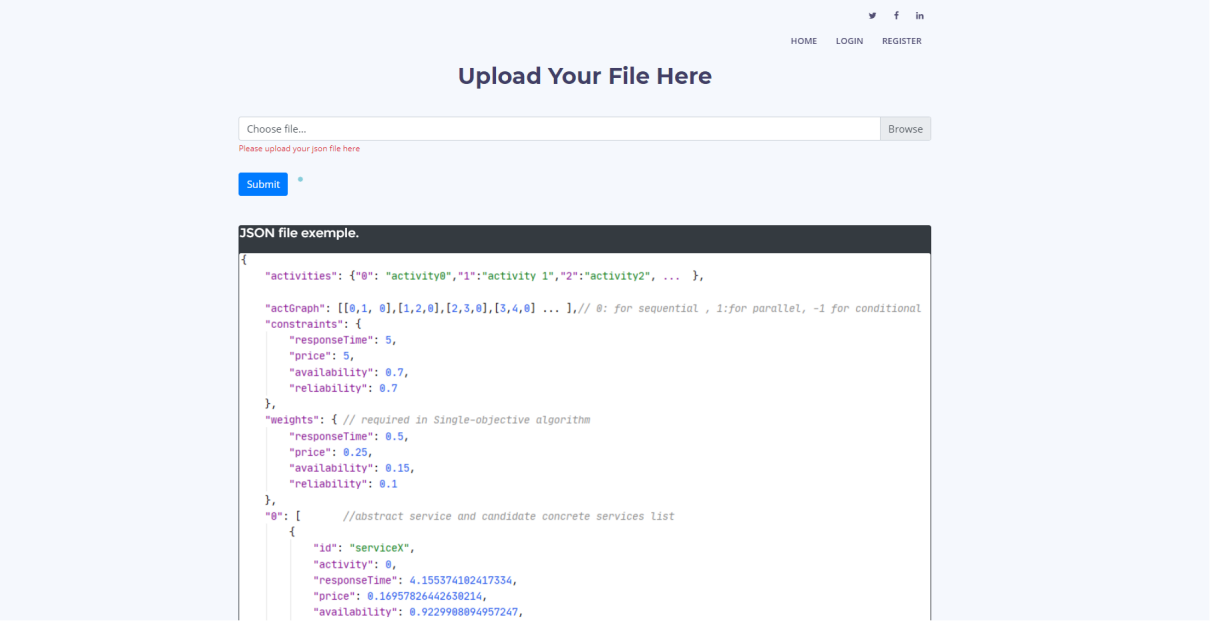


Figure 5.9: Business process submit page

The JSON file structure is explained by an example displayed beneath the submit button.Now the algorithm is executed with the preset parameters retrieved from the server.

5.1.6 Admin dashboard

The admin logs in using a specific username and password that matches an entry in the database. Once logged in, the admin can change the different parameters of the algorithms to tune their performance.

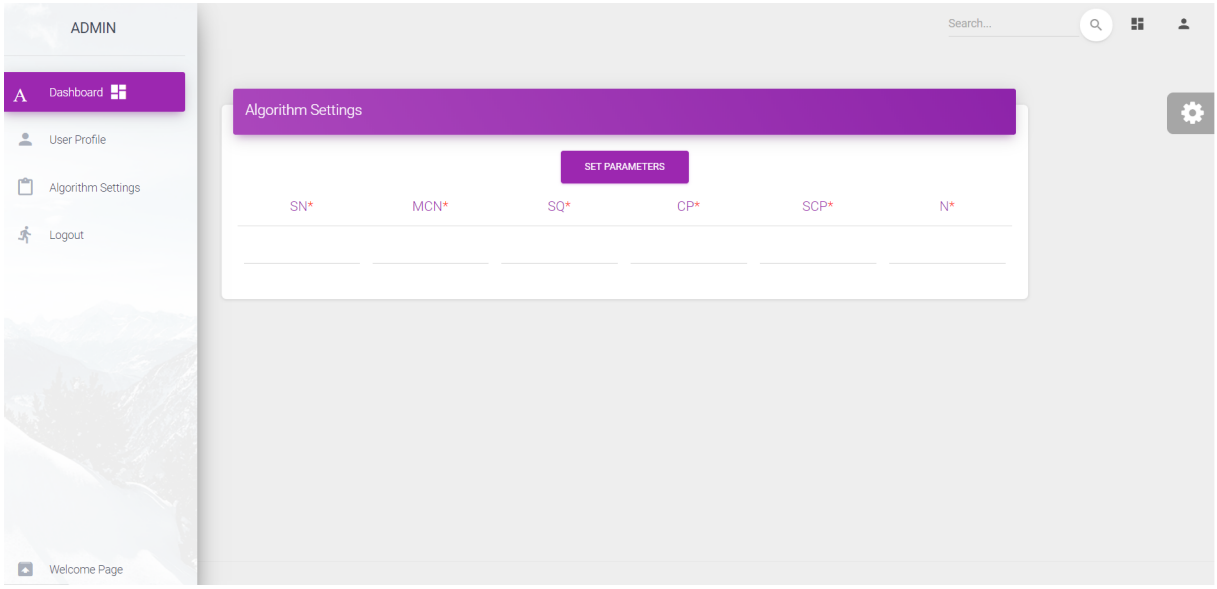


Figure 5.10: Admin dashboard

This sprint is devoted to present the complete layout of the application except the results page of the algorithm which is treated in the next sprint after implementing the first algorithm.

5.2 Sprint 2 : Hybrid ABC-GA

This sprint handles the detailed description of the ABC-GA (2.3.2), which implements the structure of the ABC algorithm (2.1.1) alongside the genetic operations (2.1.2) (mutation and uniform crossover) as a way of exploiting the solutions.

5.2.1 Design

the ABC-GA involves the same main phases of the ABC algorithm, allowing it to maintain its simplicity and ability to exploit solutions. Otherwise, it uses the genetic operators (uniform crossover, mutation), to recombine solutions via exchanging or modifying the services (genes) that constitute them. However some features in the ABC algorithms are modified such as the fitness evaluation.

Fitness evaluation

Since ABC-GA is a single-objective algorithm, a weighted sum of the criteria (Price, Response time, Availability, Reliability) represents the fitness function.(2.1) However, two of the criteria mentioned above are to be minimized, and the other two are to be maximized. Therefore we chose a unified approach of optimization (maximization) :

$$normalized(Cost^{-}) = \frac{Max - Cost^{-}}{Max - Min} \tag{5.1}$$

$$normalized(Cost^{+}) = \frac{Cost^{+} - Min}{Max - Min} \tag{5.2}$$

where $Cost^{+}$ is a QoS criteria to be maximized and, $Cost^{-}$ is a QoS criteria to be minimized
Max and Min are the maximum and minimum values of the QoS criteria.

In this case minimizing $Cost^{-}$ is similar to maximizing $normalized(Cost^{-})$, and respectively maximizing $Cost^{+}$ is similar to maximizing $normalized(Cost^{+})$.

Such normalization-formula requires recalculating the maximum and minimum values of each QoS criteria in every iteration to correctly compare the fitness value of solutions across two consecutive iterations

In addition to the fitness evaluation formula the ABC-GA introduces a new scouts behaviour at a certain late stage of the process.

Scouts behavior change

To avoid more exploration and focus on exploiting the current best solution in the final iterations of the algorithm, we introduce the notion of scouts behavior change, where scout bees start applying mutation operator on current solutions rather than searching for random solutions.A randomly chosen service from the solution that reached the exploitation limit is replaced by a randomly chosen candidate service.

This activity diagram details the different phases of our algorithm

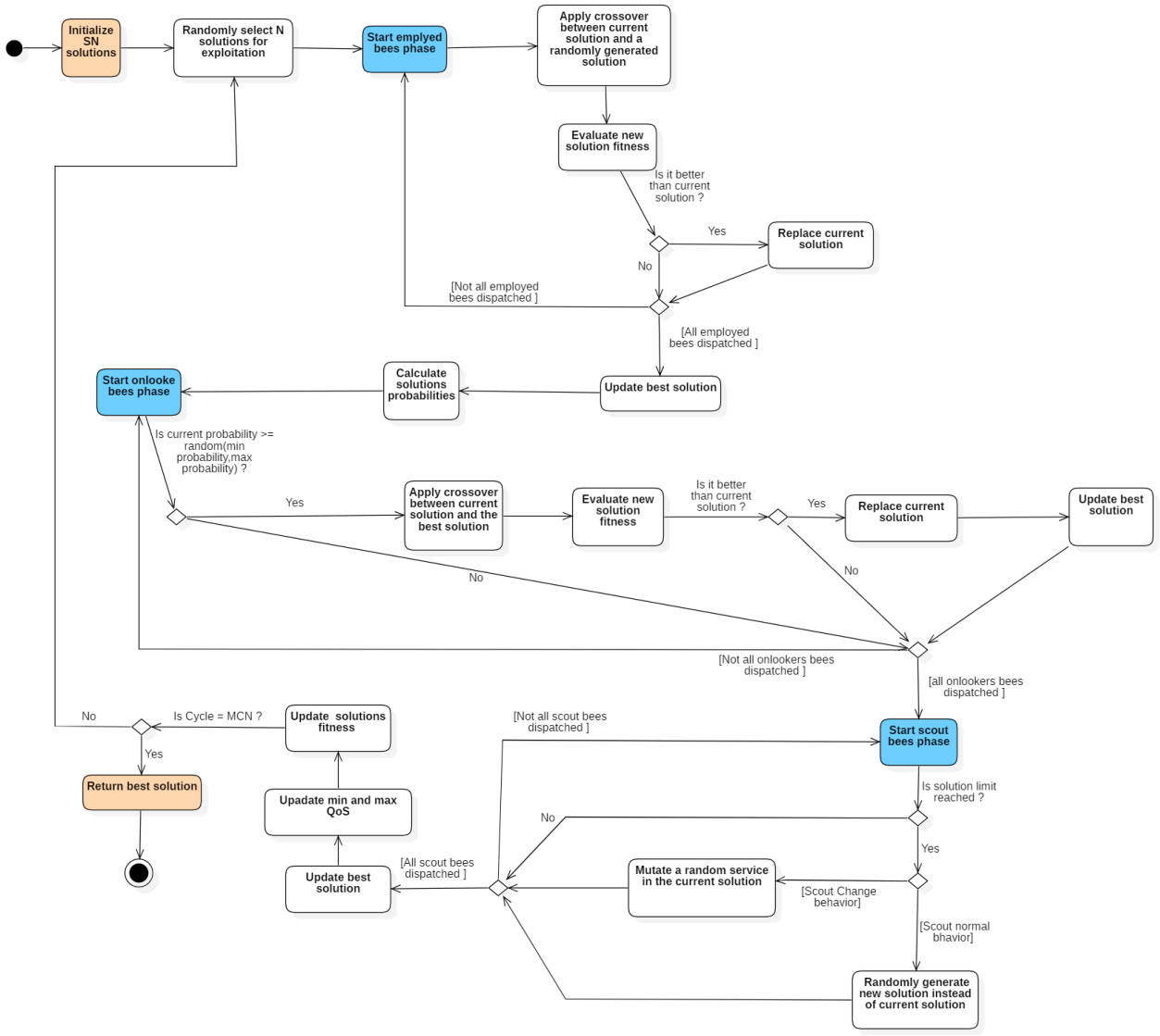


Figure 5.11: ABC-GA activity diagram

5.2.2 Results interface

The single-objective algorithm returns a single optimal solution. The different QoS of the solution are calculated, and the chosen services are displayed. Now it is up to the developer to decide whether he wants to accept the result and download it or not.

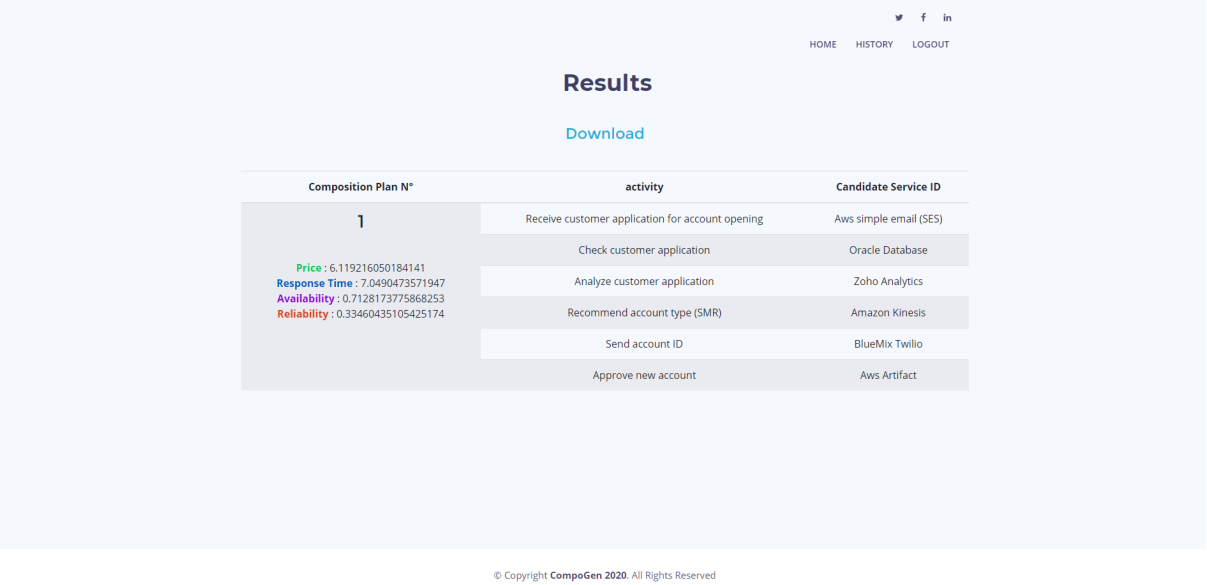


Figure 5.12: Single-objective result interface

5.2.3 Algorithm evaluation

Problem initializing

To evaluate our algorithm, we prepare examples of cloud service-composition scenarios for sequential type business plan with a various number of activities and candidate services. Candidate services are randomly created using these formulas :

- Price in $[0.1, 5]$
- Response time in $[0.1, 3]$
- Availability in $[0.9, 0.99]$
- Reliability in $[0.7, 0.95]$

Candidate services are randomly created using these formulas :

- Constraints :

  - ResponseTime: $number\ of\ activities \times 3$
  - Price: $number\ of\ activities \times 2$
  - Availability': $0.92^{number\ of\ activities}$
  - Reliability': $0.75^{number\ of\ activities}$

- weights :
  - ResponseTime: 0.1
  - Price: 0.5
  - Availability': 0.15
  - Reliability': 0.25

Execution and results

The optimal solution is determined using an exhaustive search method by executing the algorithm for 10 times the number of iterations MCN.

Each algorithm is executed 30 times and the results present an average value of the 30 execution tests

- Deviation : is the average difference between the fitness in each execution.
- Convergence : is the iteration from which the best solution no longer improves.

Algorithm	Activities	Candidates	MCN/G	SN/N	SQ	Fitness	Scalability	Deviation	Convergence
ABC-GA	**5**	**50**	**200**	**20**	**10**	**0.9989**	**2.164**	**0.0019**	**175**
ABC	**5**	**50**	**200**	**20**	**10**	**0.8616**	**1.494**	**0.0575**	**193**
GA	**5**	**50**	**200**	**20**	**-**	**0.7863**	**0.144**	**0.1030**	**96**
ABC-GA	**8**	**100**	**400**	**20**	**20**	**0.9954**	**9.774**	**0.0061**	**363**
ABC	**8**	**100**	**400**	**20**	**20**	**0.8025**	**4.069**	**0.0524**	**388**
GA	**8**	**100**	**400**	**20**	**-**	**0.6737**	**0.393**	**0.0749**	**304**
ABC-GA	**10**	**150**	**500**	**20**	**30**	**0.9686**	**22.525**	**0.0219**	**480**
ABC	**10**	**150**	**500**	**20**	**30**	**0.7975**	**7.215**	**0.0531**	**487**
GA	**10**	**150**	**500**	**20**	**-**	**0.6281**	**0.781**	**0.0789**	**420**
ABC-GA	**20**	**100**	**1000**	**30**	**100**	**0.9185**	**94.225**	**0.0286**	**978**
ABC	**20**	**100**	**1000**	**30**	**100**	**0.7919**	**28.710**	**0.0391**	**973**
GA	**20**	**100**	**1000**	**30**	**-**	**0.5405**	**3.832**	**0.0550**	**919**

Table 5.1: Single-objective comparative table

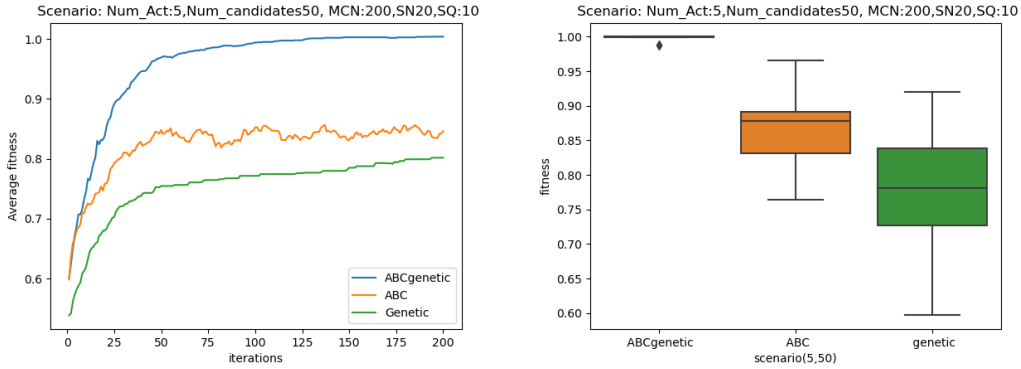


Figure 5.13: Single-objective fitness, (5,50,200,20,10) scenario plots

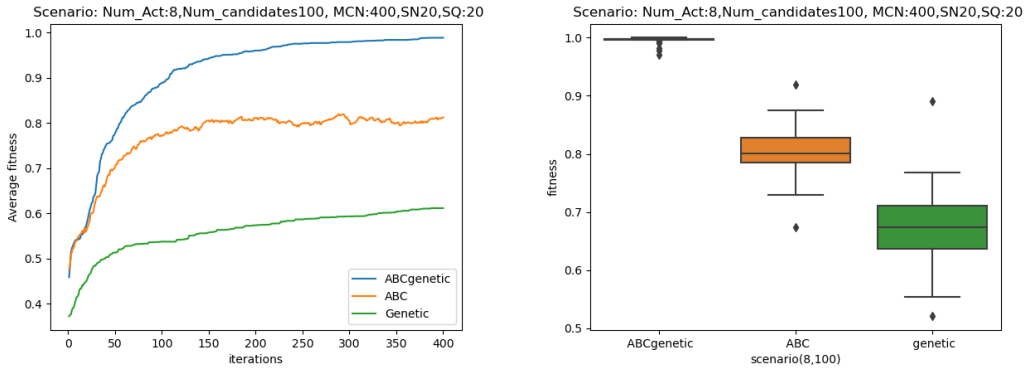


Figure 5.14: Single-objective fitness, (8,100,400,20,20) scenario plots

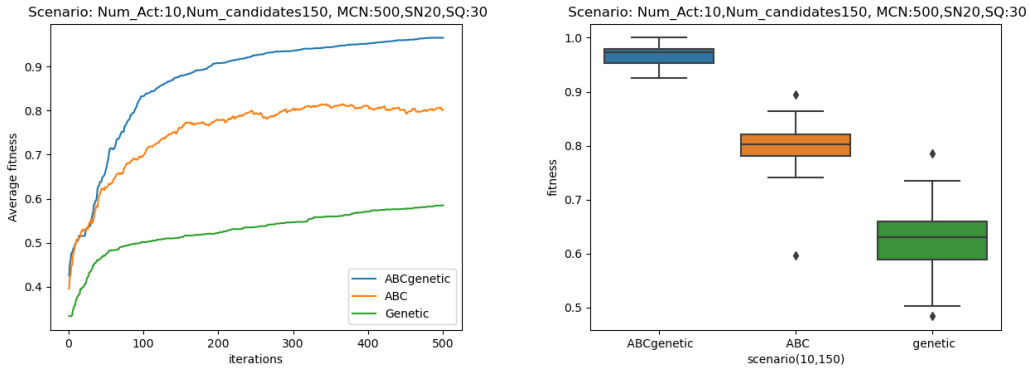


Figure 5.15: Single-objective fitness, (10,150,500,20,30) scenario plots

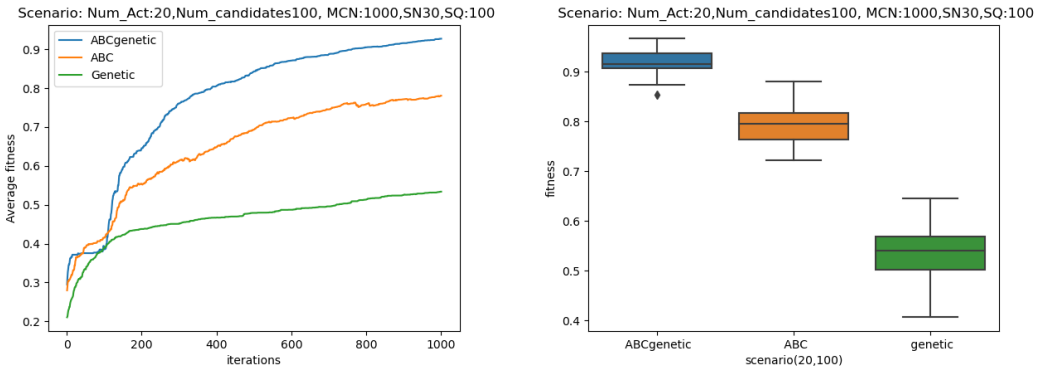


Figure 5.16: Single-objective fitness, (20,100,1000,30,100) scenario plots

Observations

As the test results show, the hybrid algorithm exceeds the ABC and GA in scalability, but such additional use of CPU resources increases significantly the fitness of the solution, which satisfies the requirements of the developer. In the next sprint, we follow the same procedure in implementing and testing the multi-objective algorithm.

5.3 Sprint 3 : Hybrid MOABC-NSGA-II

This sprint handles the detailed description of the MOABC-NSGA-II (2.3.2), which implements the structure of the MOABC algorithm (2.2.1) alongside the BSG operation from NSGA-II (2.2.2) as a way of exploiting the solutions and the non-dominated sort, crowding scores combination as a way of preserving solutions.

5.3.1 Design

The MOABC-NSGA-II, similar to the ABC-GA (inspired by the ABC), involves the same main stages of the MOABC algorithm, allowing it to maintain its simplicity and capacity to exploit solutions. Differently, it uses the genetic operators of the BSG (two-point crossover, two-way mutation), to recombine solutions via exchanging or modifying the services (genes) that constitute them. However some features in the NSGA-II algorithm are modified such as the two-way mutation.

Objective functions

In order to increase the efficiency of the algorithm by promoting the occurrences of dominating solutions we decide to consider only three objective functions to be optimized instead of four.

$$F(x) = (-f_1(x), -f_2(x), f_3(x)) \tag{5.3}$$

while $f_1(x)$, $f_2(x)$, $f_3(x)$ are respectively the response time, the price and the reliability.

The goal of this algorithm is to produce solutions that are largely spread in the search regions in order to provide various possibilities for developers with no QoS preferences.

Two-way Mutation

Because a composition plan is an aggregate of services, not a representation of 0s and 1s, the two-way mutation is transformed into two random mutations that generate two different offsprings.

This activity diagram details the different phases of our algorithm :

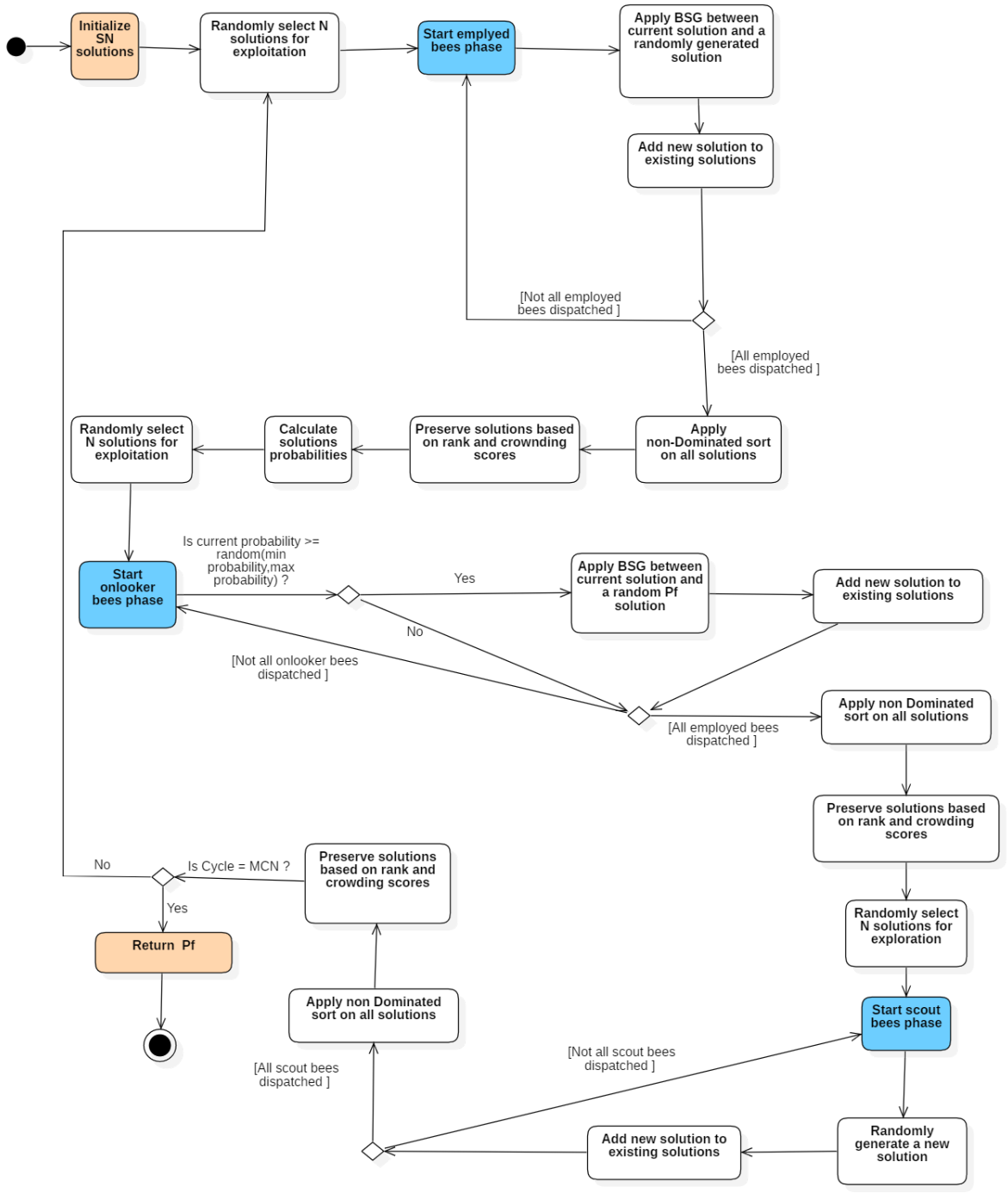


Figure 5.17: MOABC-NSGA-II activity diagram

5.3.2 Results interface

The multi-objective algorithm returns a set of non dominated solutions. The different QoS of the solution are calculated, and the chosen services are displayed. Now it is up to the developer to decide whether he wants to accept the result and download it or not.

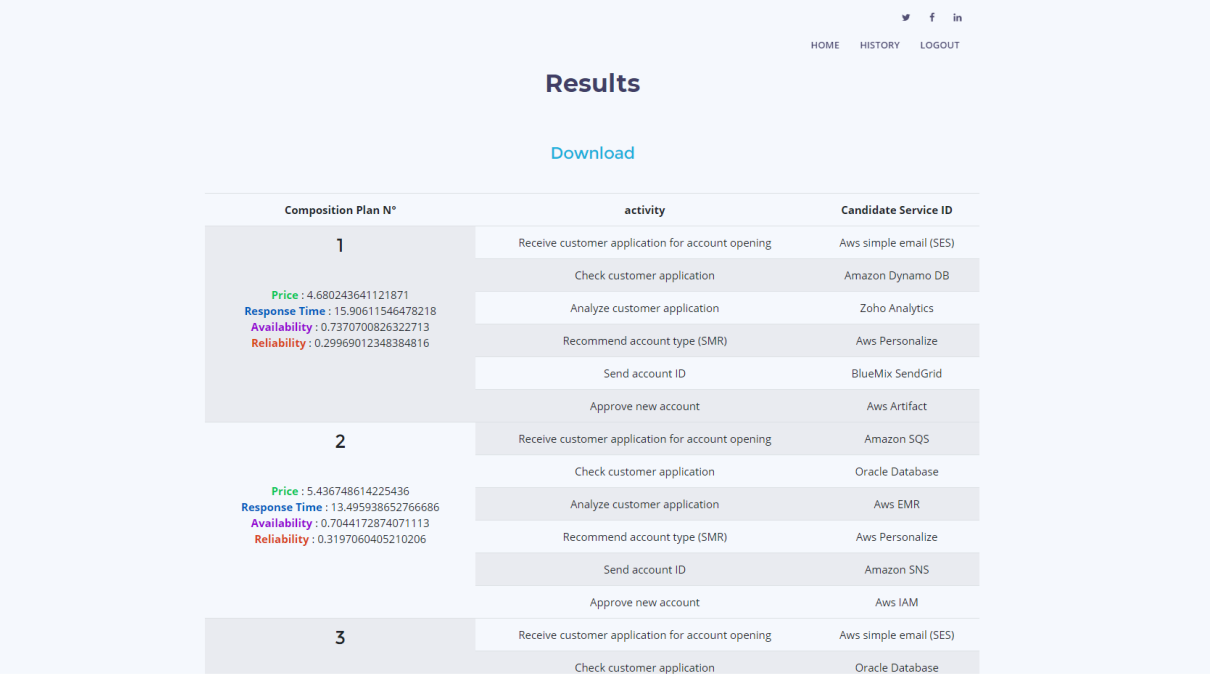


Figure 5.18: Multi-objective result interface

5.3.3 Algorithm evaluation

Problem initializing

The problem initializing is similar to the ABC-GA using the same constraints and candidate services.

Execution and results

The true Pareto is a set of Pareto front solutions produced by each algorithm in 10 executions, on which we apply the non dominated sorting, resulting in a non-dominated ("optimal") solutions.

Each algorithm is executed 30 times and the results present an average value of the 30 execution tests.

In order to evaluate our algorithm, we present three evaluation metrics used for the MOO algorithms:

- GD : This indicator measures the closeness of the solutions to the true Pareto front.

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{(n)} \tag{5.4}$$

  where $d_i = \min \|f(x_i) - PF_{\text{true}}(x_i)\|$ refers to the distance in objective space between individual x_i and the nearest member in the true Pareto front, and n is the number of individuals in the approximation front.

- IGD: This metric measures both convergence and diversity.

$$IGD = \frac{\sum_{v \in PF_{\text{tne}}} d(\nu, X)}{(PF_{\text{true}})}$$

  $d(v, X)$ denotes the minimum Euclidean distance between v and the points in X. To have a low value of IGD, the set X should be close to PF_{true} and cannot miss any part of the whole $\text{PF}_{\text{truc.}}$ The only difference between GD and IGD is that in the later we don't miss any part of the true Pareto Front set

- HV: This metric measures the volume of the objective space between the obtained solutions set and a specified reference point in the objective space. For HV, a larger value is preferable.
  HV is interesting because it captures both the proximity of the approximation solution set (X) to the true Pareto Front (PF) and the distribution of F over the objective space.

Algorithm	Activities	Candidates	MCN/G	SN/N	GD	IGD	HV	Scalability
NSGA-II	**5**	**50**	**200**	**50**	**0.0313**	**0.0207**	**0.5016**	**10.570**
MOABC	**5**	**50**	**200**	**50**	**0.0101**	**0.0132**	**0.6979**	**43.660**
MOABC-NSGA-II	**5**	**50**	**200**	**50**	**0.0102**	**0.0078**	**0.7185**	**118.337**
NSGA-II	**10**	**100**	**500**	**50**	**0.0297**	**0.0286**	**0.3536**	**31.722**
MOABC	**10**	**100**	**500**	**50**	**0.0132**	**0.0156**	**0.5620**	**101.922**
MOABC-NSGA-II	**10**	**100**	**500**	**50**	**0.0151**	**0.0146**	**0.5843**	**338.034**
NSGA-II	**20**	**20**	**1000**	**50**	**0.0196**	**0.0211**	**0.2374**	**68.052**
MOABC	**20**	**20**	**1000**	**50**	**0.0134**	**0.0140**	**0.2997**	**227.181**
MOABC-NSGA-II	**20**	**20**	**1000**	**50**	**0.0152**	**0.0117**	**0.3132**	**607.703**

Table 5.2: Multi-objective comparative table

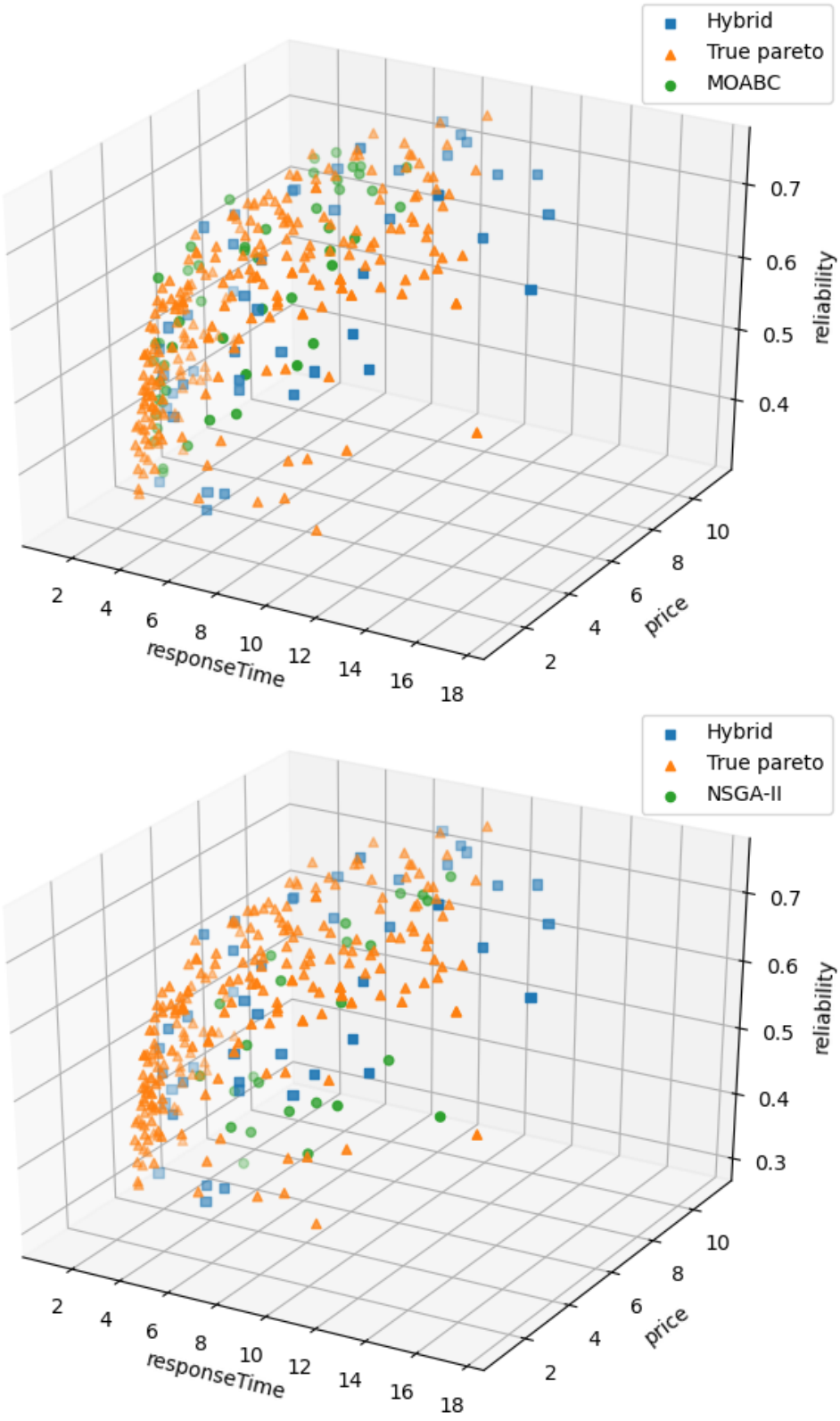


Figure 5.19: Multi-objective solutions, (5,50,200,50,10) scenario plots

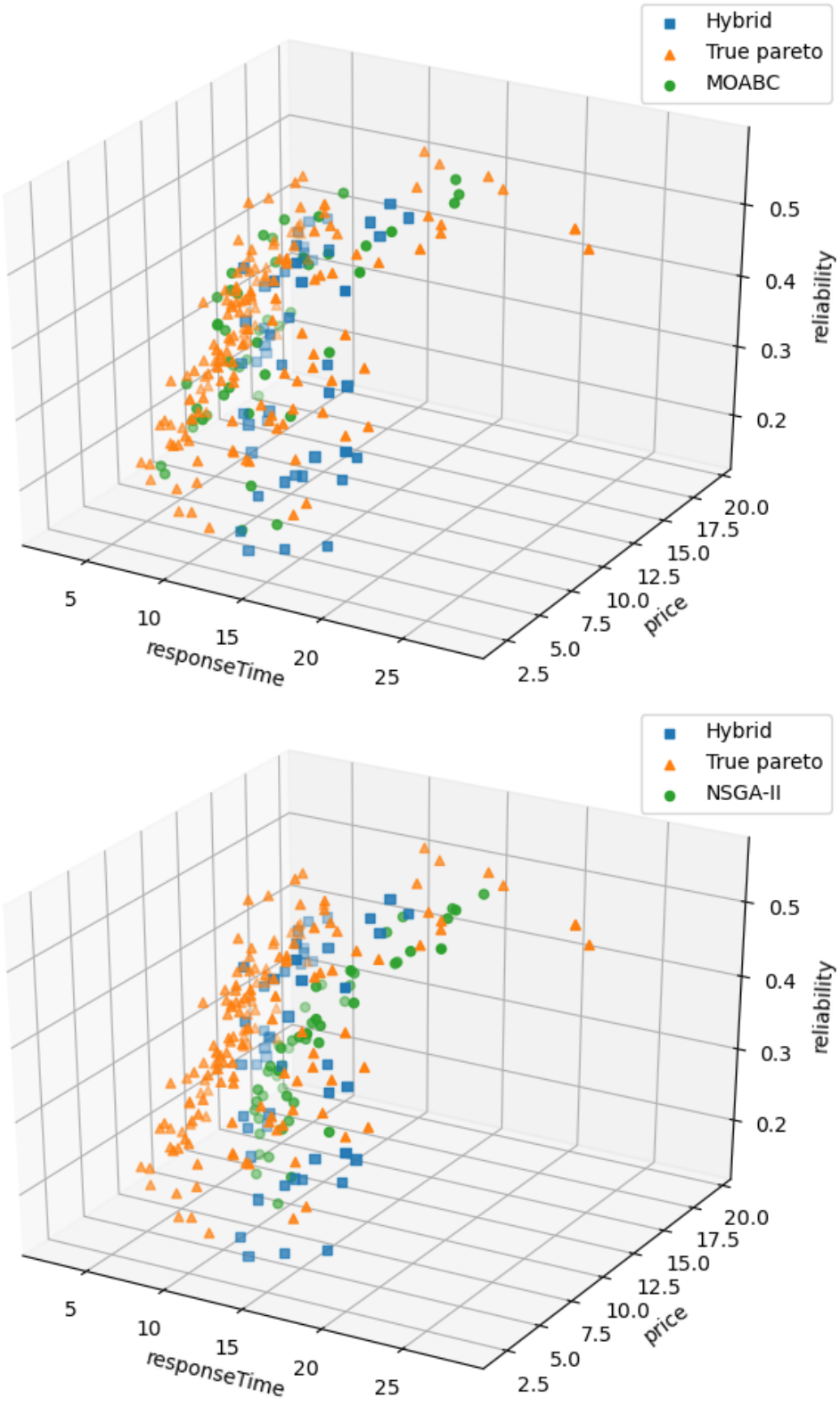


Figure 5.20: Multi-objective solutions, (10,100,500,50,20) scenario plots

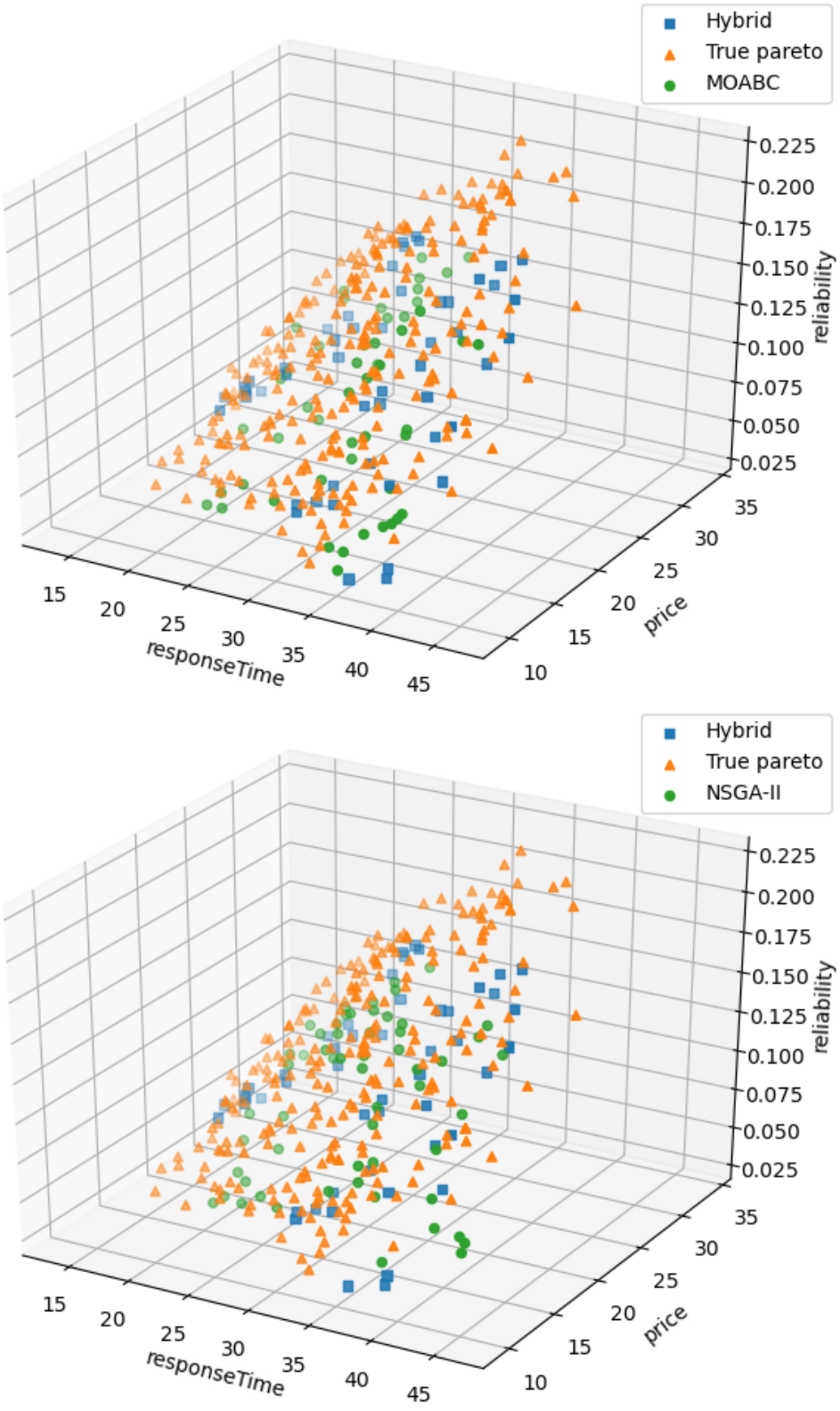


Figure 5.21: Multi-objective solutions, (20,20,1000,50,50) scenario plots

Observations

As the test results show, the hybrid algorithm exceeds the MOABC and NSGA-II in scalability, but such additional use of CPU resources increases both the distribution of the solutions and their closeness to the true Pareto.

It is also noticeable that the hybrid algorithm is just slightly better than the MOABC algorithm, but that is only the result of the MOABC algorithm implementation which is based on discrete mutations instead of the continuous approach.

Conclusion

This chapter unveils a full description of all the components that constitute our application from the development of the different platform pages to the back-end optimization hybrid algorithms and their performance evaluation.

General conclusion

Cloud computing is a computing paradigm where infrastructure, platforms, and software are provided as services, which lead to raising the number of services available on the market. Many services offered the same functionalities. Therefore, the user has a multitude of choices for one specific functionality. On the other hand, in most cases, the user's request involves many services and not just one. Hence, these services must be suitably chosen to build a global service that satisfies the user's demand.

This report is a brief project synthesis that describes in detail the implementation of a web application that builds a suitable cloud composition plan meeting the developer's requirements.

To achieve this we follow an outlined plan: First, we introduce the main concepts underlying our project, and we determine the problem that we are facing. Second, we establish a theoretical study of the existing solutions and note their major privileges and limitations. Thus, we propose a solution which is both context satisfactory and inspired by the state of the art. Third, we align the global design and architecture with the specified requirements of the developer, which enables us to assess the tasks that must be achieved, the time they require, and their order. Finally, we implement the web platform and its diverse features, and we conclude our project by implementing the back-end algorithms and evaluating them.

It is essential to state that, achieving this project was a delicate process, we encountered during our work many setbacks on both the theoretical and applied basis. However, each difficulty presents a great opportunity to sharpen our minds and magnify our analytical abilities becoming more familiar with optimization algorithms, and more experienced in developing flexible and maintainable software.

As every project can be more improved and elaborated, various aspects of our project can be enhanced. Perhaps the most important feature that can be added is the prediction of suitable parameters for our algorithms, depending on the input of the developer. Because modifying the parameters has a huge effect on the performance of the algorithms, preset parameters can never work efficiently for all sorts of business processes. An intelligent statistical prediction presents a huge contribution to our project.

Bibliography

[1] Jiajun Zhou and Xifan Yao. A hybrid artificial bee colony algorithm for optimal selection of qos-based cloud manufacturing service composition. *Springer-Verlag London*, page 3374, 2016.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transaction on Evolutionary Computation, Vol. 6, NO. 2*, 2002.

[3] H. Gabsi, R. Drira, H. B. Ghezala, and S. Ducasse. From business process to cloud application. *International Business Information Management Conference (35th IBIMA)Seville, Spain*, April 2020.

[4] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, Juan F Pérez, and Weikun Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications volume*, 2014.

[5] Wang D, Yang Y, and Mi Z. A genetic-based approach to web service composition in geo-distributed cloud environment. *Comput Electr Eng 43*, page 129–141, 2015.

[6] João M.P.Cardoso, José Gabriel F.Coutinho, and Pedro C.Diniz. *Embedded Computing for High Performance: Efficient Mapping of Computations Using Customization, Code Transformations and Compilation.* 2017.

[7] D Karaboga. An idea based on honey bee swarm for numerical optimization. *Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department*, 2005.

[8] B. Karaboga, D.and Basturk. On the performance of artificial bee colony (abc) algorithm. *Appl. Soft Comput. 8 (1)*, page 687–697, 2008.

[9] J.H. Holland. *Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan.* MIT Press (1992), 1975.

[10] Emrah, Bing Xue, Mengjie Zhang, Dervis Karaboga, and Bahriye Akay. Pareto front feature selection based on artificial bee colony optimization. *journal of Information Sciences*, 2017.

[11] Yusliza Yusoff, Mohd Salihin Ngadiman, and Azlan Mohd Zain. Overview of nsga-ii for optimizing machining process parameters. *Procedia Engineering Vol. 15*, pages 3978–3983, 2011.

[12] Eduardo Gerhardt and Herbert Martins Gomes. Artificial bee colony (abc) algorithm for engineering optimization problems.

[13] Xin-She Yang. *Nature-Inspired Optimization Algorithms*. 2014.

Netography

[14] https://brainhub.eu/blog/differences-lean-agile-scrum/, 18/06/2020.

[15] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3, 18/06/2020.

[16] https://www.slideshare.net/JeremyFisher1/genetic-algorithms-programming-by-the-seat-of-your-genes, 18/06/2020.

[17] https://www.researchgate.net/figure/The-drawback-of-the-crowding-distance-based-sorting_fig4_271730329, 18/06/2020.

[18] https://i.ytimg.com/vi/SL-u_7hIqjA/maxresdefault.jpg, 18/06/2020.

[19] https://www.citrix.com/glossary/what-is-a-cloud-service.html, 18/06/2020.

[20] http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm, 18/06/2020.