5-31-2017

Evolutionary Game Theoretic Multi-Objective Optimization Algorithms and Their Applications

Yi Ren Cheng
University of Massachusetts Boston

EVOLUTIONARY GAME THEORETIC MULTI-OBJECTIVE OPTIMIZATION ALGORITHMS AND THEIR APPLICATIONS

A Dissertation Presented

by

YI REN CHENG

Submitted to the Office of Graduate Studies,
University of Massachusetts Boston,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

May 2017

Computer Science Program

EVOLUTIONARY GAME THEORETIC MULTI-OBJECTIVE OPTIMIZATION ALGORITHMS AND THEIR APPLICATIONS

A Dissertation Presented

by

YI REN CHENG

Approved as to style and content by:

Junichi Suzuki, Associate Professor
Chairperson of Committee

Dan A. Simovici, Professor
Member

Ding Wei, Associate Professor
Member

Alfred G. Noel, Professor
Member

Dan A. Simovici, Program Director
Computer Science Program

Peter Fejer, Chairperson
Computer Science Department

ABSTRACT

EVOLUTIONARY GAME THEORETIC MULTI-OBJECTIVE OPTIMIZATION ALGORITHMS AND THEIR APPLICATIONS

May 2017

Yi Ren Cheng,
B.A., University of Ramon Llull, Spain
M.E., University of Ramon Llull, Spain
M.S., University of Massachusetts Boston
Ph.D., University of Massachusetts Boston

Directed by Associate Professor Junichi Suzuki

Multi-objective optimization problems require more than one objective functions to be optimized simultaneously. They are widely applied in many science fields, including engineering, economics and logistics where optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives. Most of the real world multi-objective optimization problems are NP-Hard problems. It may be too computationally costly to find an exact solution but sometimes a near optimal solution is sufficient. In these cases Multi-Objective Evolutionary Algorithms (MOEAs) provide good approximate solutions to problems that cannot be solved easily using other techniques. However Evolutionary Algorithm is not stable due to its random nature, it may produces very different results every time it runs. This dissertation proposes an Evolutionary Game Theory (EGT) framework based algorithm (EGTMOA) that provides optimality and stability at the same time. EGTMOA combines the notion of stability from EGT and optimality from MOEA

to form a novel and promising algorithm to solve multi-objective optimization problems. This dissertation studies three different multi-objective optimization applications, Cloud Virtual Machine Placement, Body Sensor Networks, and Multi-Hub Molecular Communication along with their proposed EGTMOA framework based algorithms. Experiment results show that EGTMOAs outperform many well known multi-objective evolutionary algorithms in stability, performance and runtime.

ACKNOWLEDGEMENTS

It is a long journey to complete this dissertation. I can not accomplish it without all your helps. Here, I would like to deliver my most sincerely gratitude to the following.

My advisor Prof. Junichi Suzuki, for the continuous support of my Ph.D study and related research, for his patience, motivation, guidance and immense knowledge.

Dr. Dan Simovici, Dr. Wei Ding, and Dr. Alfred Noel for their insightful comments and encouragement. And thank you for being part of my thesis committee.

Computer Science Department faculty, for their tutoring with excellent courses and for providing useful academic resources. Computer Science staff, for their great help in my academic administration and guidance.

My friends Thamer Altuwaiyan, Nada Attar, Dung Phan, Ting Zhang, Quynh Vo, Tong Wang, Kaixun Hua for their accompany, friendship and support through all my academic years in UMASS Boston.

A big thanks to my parents (Jianwei Ren and Xinguang Cheng) and my parents in law (Guanghui Xiao and Hong Wang), for their giving love without any returns and for their constantly support and guidance in my entire life.

My daughter Jana Ren, for well behaved and being a good kid supporting her father during all these time.

In the end I would like to specially thanks my wife Wen Xiao Ren, for being an excellent life partner, a responsible and patient mother, a lovely and brilliant wife. Your love is the fuel that allows me to do the impossible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Many real world problems are often require to satisfy multiple criterion at the same time. Those problems are called multi-objective optimization problems where more than one objective are presented and need to be optimized simultaneously. We could find those kind of problems everywhere in the world, and anytime around our ordinary life. For instance buying a used car we may consider minimizing the cost and mileage while maximizing the MPH (mileage per hour), or designing an aircraft that need to satisfy hundreds of criteria (speed, capacity, energy consumption, acquisition cost, assembly hours, etc...).

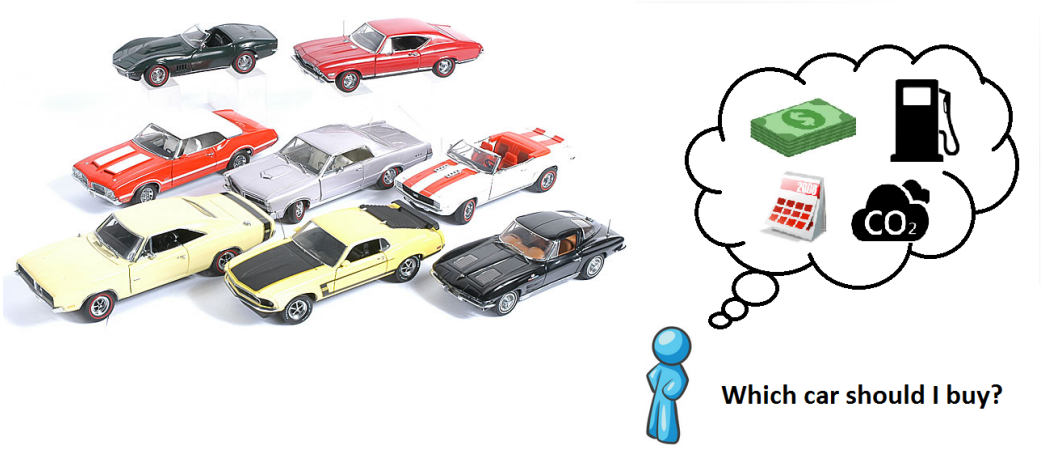


Figure 1: A multi-objective optimization problem: buying a used car

Multi-objective optimization problems often deal with multiple conflicting objectives and may subject to many constraints. In the case of buying a used car Fig. 1, we could have several constraints such as our budget, acceptable mileage, manufacture years, car brand and model, so on. Due to its computational complexity and the huge solution space presented, it has always been a challenge solving multi-objective optimization problems.

In fact, most of real-life multi-objective optimization problems are often of exponential size, a straightforward reduction from the knapsack problem shows that they are NP-hard to compute. Thus, it is computationally too costly to find an exact optimal solution if one exists. And most of the time in real applications it is quite hard for the decision maker to have all the information to correctly and completely formulate them. Therefore, in such situations finding a near optimal solution is a practical approach that fits into multi-objective optimization problems.

The set of all feasible solutions is called Pareto curve or surface Fig. 2 which represents the solution space of the multi-objective optimization problem. Due to conflicting objectives and constraints multi-objective optimization problems lead to not a single optimal solution, but a set of non-dominated solutions which is called Pareto front Fig. 2. More details about Pareto optimal solutions will be given in section 2.1.

Numerous researches have been studied with many different methods proposed to solve multi objective optimization problems. In the next section we are going to review a little what historically have done and what the current state of the art is in question.

1.1 Related works

Many techniques to solve multi-objective optimization problems were proposed in the past. In the following we will start revising some of the most relevant techniques described in [Car08].

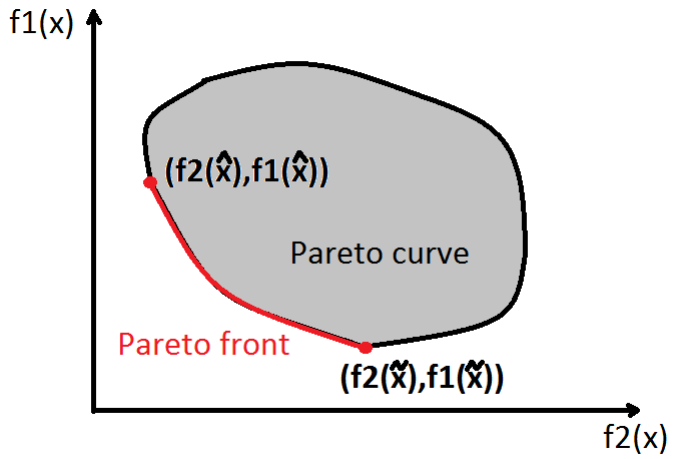


Figure 2: Example of a Pareto curve and Pareto front of two objective functions

- The Weighted-sum method

  The basic idea is to combine multiple objectives into one single-objective scalar function in order to solve a multi-objective problem. This approach is also know as weighted-sum or scalarization method. The goal is to minimize a positively weighted sum of the objectives.

$$min \sum_{i=1}^{n} \gamma_i \cdot f_i(x)$$

$$\sum_{i=1}^{n} \gamma_i = 1$$

$$\gamma_i > 0, i = 1, ..., n$$

$$x \in S$$

  After combining multiple objectives into one single objective we have a new optimization problem with an unique objective function. Two main drawbacks are presented in this approach. The first one is the possibly huge computation time involved

by considering different weight values. And when the Pareto curve is non-convex Fig. 3 there is a set of points that cannot be reached for any combination of the weight vector.

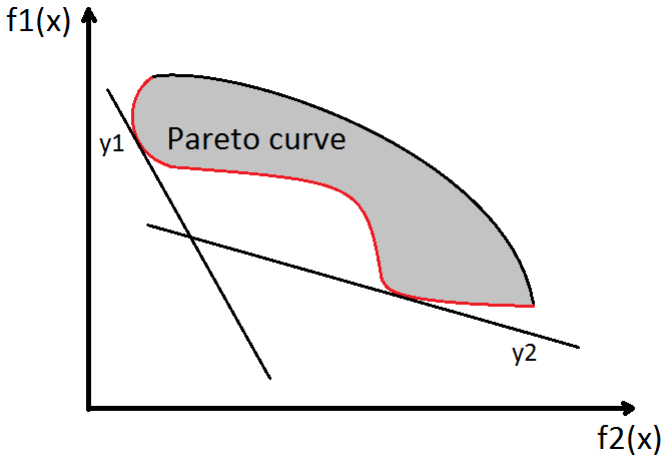


Figure 3: Geometrical representation of the weight-sum approach in the non-convex Pareto curve case

- ε-constraints Method

  The decision maker chooses one objective out of n to be minimized, and the remaining objectives are constrained to be less than or equal to given target values.

$$min f_j(x)$$

$$f_i(x) \leq \varepsilon_i, \forall_i \in \{1,...,n\} \setminus \{j\}$$

$$x \in S$$

  One advantage of the ε-constraints method is that it is able to achieve efficient points in a non-convex Pareto curve. In Fig. 4, when $f_2(x) = \varepsilon_2, f_1(x)$ is an efficient point of the non-convex Pareto curve.

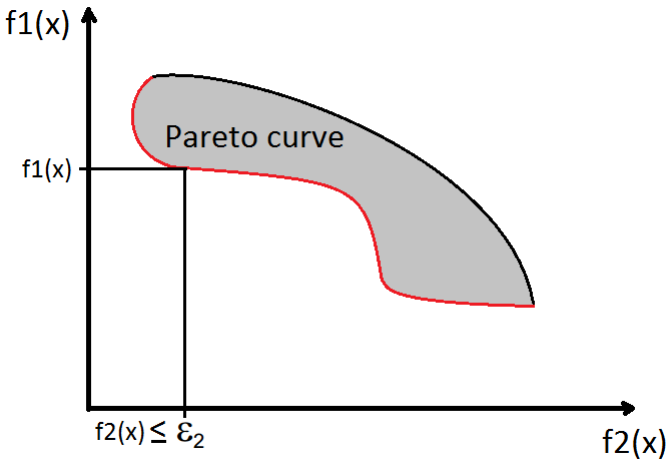


Figure 4: Geometrical representation of the ε-constraints approach in the non-convex Pareto curve case

The drawback of this method is that the decision maker has to choose appropriate upper bounds ε_i values for the constraints. Moreover, the method is not efficient if the number of objective functions is greater than two.

- Multi-level Programming

  Multi-level programming aims to find one optimal point in the entire Pareto surface. Multi-level programming optimizes the n objectives in a predefined order. It firstly minimizes the first objective function, and then it searches for minimizing the second most important objective, and so on until all the objective function have been optimized.

  It works if the order among objectives is meaningful and user is not interested in the continuous trade off among the functions. The main drawback is that the less important objective functions tend to have no influence on the overall optimal solution.

- Goal Programming

Goal Programming attempts to find specific meta values of these objectives. In fact, it does not solve directly a multiple objectives optimization problem, it tries to find a solution that accomplishes a specific goal. An example is shown below.

$$f_1(x) \geq v_1$$

$$f_2(x) = v_2$$

$$f_3(x) \leq v_3$$

$$x \in S$$

- Evolutionary Algorithm

  Most recent studies focus on evolutionary algorithms (EA) which shown to be a promising method solving multi-objective optimization problems with conflicting objectives by approximating the Pareto solution set. EA is inspired by biological evolution, it uses biologic mechanisms such as reproduction, mutation, crossover, and selection. More details will be given later in section 2.2.

  The main advantage is that EAs are metaheuristic algorithms, they do not make any assumption about the underlying fitness landscape. Therefore EA often perform well approximating solutions to all types of problems in many diverse fields as engineering, biology, economics, marketing, social sciences, so on.

  Some of the most well known EAs are

  - Genetic Algorithm: Probably this is the most popular type of EA. GAs are commonly used to generate high quality solution set by relying on biological inspired operators such as mutation, crossover and selection. Non-dominated Sorting Genetic Algorithm - II known as NSGA-II and Strength Pareto Evolu-

tionary Algorithm 2 also known as SPEA-2 are well known variants of GA that have become as GA standard approaches.

- Differential Evolution: DE optimizes a problem by maintaining a population of candidate solutions and to create new candidate solutions by combining existing ones using a differential equation. And then keeping the solution with the best fitness value on the optimization problem at hand.

The main drawback of EA is that it relies heavily on stochastic mechanism, due to its random nature EA is highly unstable in general. Unstable here means under the same problem setting EA could give a very different performance result for each run.

1.2 Contributions

The main goal of this dissertation is to propose a novel algorithm Evolutionary Game Theoretic Multi-Objective Algorithms (EGTMOA) Chapter 3 that aims to solve multi-objective optimization problems considering stability, optimality and running time. EGTMOA is an Evolutionary Game Theory framework based Evolutionary Algorithm. It combines the stability property from EGT and the optimality notion from EA together to form a new type of metaheuristic algorithm that guarantees to deliver a stable and high quality solution in a reasonable running time. Main contributions are listed as follow.

1. Evolutionary Game Theoretic Multi-Objective Algorithms (EGTMOA): a new metaheuristic algorithm framework EGTMOA is proposed to solve multi-objective optimization problems in a stable, optimal and fast manner.

2. Cloud Virtual Machine Deployment: Formulation of a new multi-objective optimization problem with four objectives and four constraints that is designed to describe the resource allocation problem in a Cloud Data Center.

3. Cielo, AGEGT, and Cielo-LP: Description of three EGTMOA framework based algorithms that are aimed to solve the formulated Cloud Virtual Machine Deployment multi-objective optimization problem. Their evaluation are performed and studied through different experiments.

4. Body Sensor Network: Creating a new multi-objective optimization problem that attempting to formulate a constrained data transmitting scheduling problem for in-body sensor networks environment

5. BitC: Another EGTMOA framework based algorithm that is designed to solve the Body Sensor Network problem. Evaluation of EGTMOA is performed and studied through different experiments.

6. Molecular Communication: It simulates an in-body Multi-Hub Molecular Communication environment, and formulates a new non-constrained two objective optimization problem to improve its communication performance and efficiency

7. EMMCO: A variant of EGTMOA algorithm that is proposed to solve the Multi-Hub Molecular Communication problem. Evaluation of EMMCO is performed and studied through different experiments.

1.3 Workflow

The rest of dissertation is organized as follow: Chapter 2 provides an overview of related concepts that lays foundation for this dissertation. Chapter 3 gives in detail all the components of the proposed approach EGTMOA. In Chapter 4, 5 and 6 I describe three different multi-objective optimization applications, Cloud Virtual Machine Deployment, Body Sensor Network, and Multi-Hub Molecular Communication with their respective proposed EGTMOA framework based algorithms and experiment evaluations. Chapter 7 concludes

the main contributions of this dissertation, and Chapter 8 discusses potential future research directions originated from this thesis.

CHAPTER 2

BACKGROUND

2.1 Multi-Objective Optimization

Many classical optimization problem consists of optimizing a single objective, for instance finding the shortest path from an origin to a destination in a network is one of the most classical optimization problems in transportation and logistic. However most of real-life problems in nature have several and possibly conflicting objectives to be satisfied simultaneously. Fig. 5 shows an example of a multi-objective optimization problem with two conflicting objectives, where we try to find an optimal solution that minimizing the cost of a data center while reducing its data transmission latency. Here in this example the cost could be money that is spent on Internet connection, data center power consumption, hardware equipments, so on. And it is not hard to conclude that more money we spent better equipments and connectivity we have, thus as consequence faster the data transmission is and less latency we could have. Since our objective is to minimize the cost and latency, it is clear that these two objectives are conflicting with each other. The utopia point represents the best ideal solution to the formulated problem that usually is not possible to be reached. In this case it is $(0, 0)$ which means it costs 0$ to have a latency of $0ms$, and of course this is not possible.

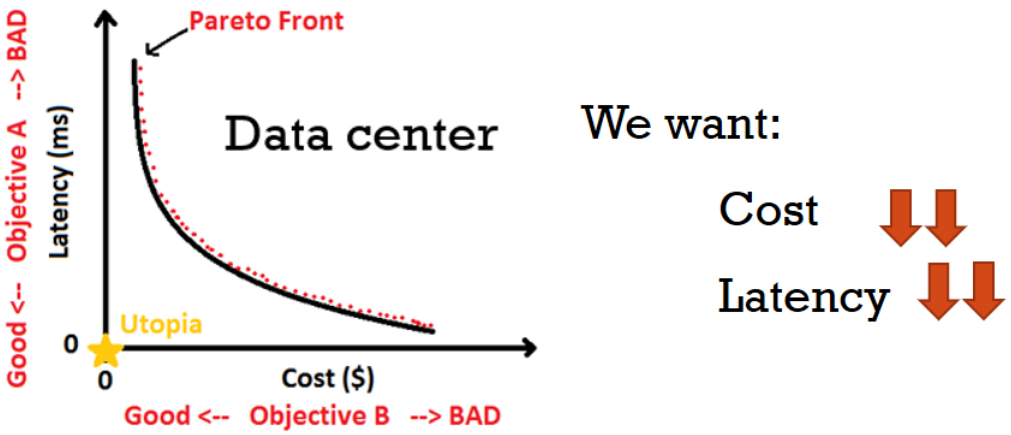


Figure 5: An example of multi-objective optimization problem with two conflicting objectives

In mathematical terms, a multi-objective optimization problem can be formulated as follow.

$$min(f_1(x), f_2(x), ..., f_k(x))$$

$$\text{s.t. } x \in \mathbb{X}$$

Where $x \in \mathbb{R}^n$ is a vector of n decision variables which represent the values to be chosen in the optimization problem. $\mathbb{X} \subseteq \mathbb{R}^n$ denotes the feasible set that is implicitly determined by a set of equality and inequality constraints. $f : \mathbb{R}^n \to \mathbb{R}^k$ is a vector of k objective functions that map n decision variables to k objective values Fig. 6. In multi-objective optimization, the sets $\mathbb{R}^n$ and $\mathbb{R}^k$ are known as decision variable space and objective function space respectively.

The ultimate goal in a multi-objective optimization problem is to find an optimal solution that satisfy simultaneously all the objectives, however multi-objective optimization problems often do not exist a single solution that optimizes each objective at the same time

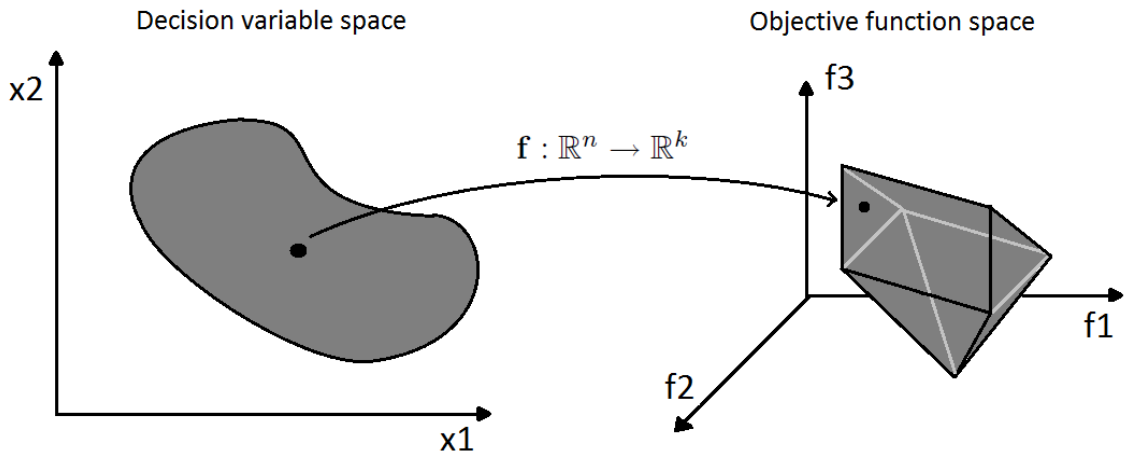


Figure 6: Search spaces in multi-objective optimization problems

because of conflicting objectives. In this case, there exists a number of Pareto optimal solutions.

2.1.1 Pareto Optimal Solutions

The set of all feasible solutions is called Pareto curve or surface Fig. 2 which represents the solution space of the multi-objective optimization problem. A solution is called Pareto optimal, if none of the objective functions can be improved in value without degrading some of the other objective values. And the set of Pareto optimal solutions forms Pareto front, where all solution are non-dominated with each other.

Considering the data center example in Fig. 5 let's give three solutions with respective objective values $\{S_1 = (30ms, 420K\$), S_2 = (15ms, 380K\$), S3 = (10ms, 400K\$)\}$ Fig. 7. It is clear that S_2 outperforms S_1 in both objectives latency and cost. However S_2 is better than S_3 in cost objective value, but worst in latency. Therefore S_2 and S_3 are non-dominated with each other, and they both belong to the Pareto front set.

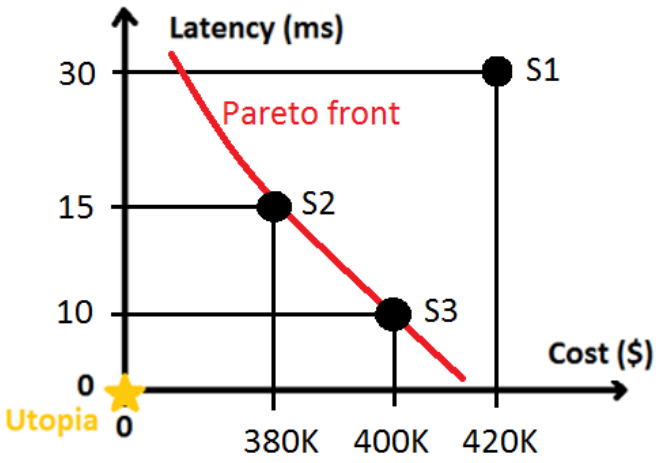


Figure 7: Three Pareto solutions for data center problem

2.2 Multi-Objective Evolutionary Algorithms

Many real world multi-objective optimization problem are NP-Hard Problems. It is not feasible to use brute force search for solving those problems due to the huge amount of computation involved. It is too computationally costly to find an exact solution but sometimes a near optimal solution is sufficient. In these cases MOEAs provide good approximate solutions to problems that cannot be solved easily using other techniques. EA uses biological evolution inspired mechanisms, such as mutation, crossover, and selection. Fig. 8 shows a concise work flow of Evolutionary Algorithm where X_n is the vector of decision variables and Y_m is the vector of objectives.

MOEAs are stochastic search and optimization methods that lead a population of candidate solutions toward the Pareto front through evolutionary mechanism and biological operators. Algorithm 2.2.1 shows steps that a traditional EA follows. Candidate solutions also called individuals in EA term are set of different decision variable vectors that are randomly generated initially (Line 2). EAs uses mutation and crossover operators to gen-

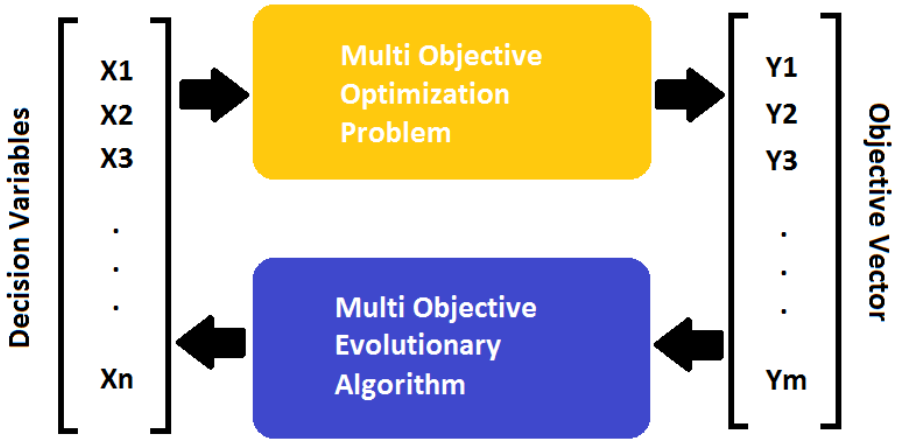


Figure 8: A concise work flow of Evolutionary Algorithm

erate new offspring solutions from the original population (Line 5). Later these solutions are evaluated by fitness functions which are objective functions defined by the optimization problem (Line 6). And based on the quality or fitness values of solutions EAs uses selection operator to chose the best non-dominated solution set to form a new population (Line 7). This whole process iterates many times or generations until it satisfies the termination condition.

MOEAs presents many advantages over traditional multi-objective approaches:

- MOEAs attempts to search the whole Pareto front instead of one single Pareto optimal solution in each run.
- MOEAs do not require any domain knowledge about the problem to be solved
- MOEAs do not make any assumption about the Pareto curve.

MOEAs do not guarantee to find the true Pareto optimal set, but instead aim to generate a good approximation of such set in a reasonable computational time. The main drawback of EA is the lack of stability. Due to its stochastic mechanism, EA is not stable. Stability

Algorithm 2.2.1: Evolutionary Algorithm Pseudo-code

1: $t \leftarrow 0$;
2: $InitPopulation[P(t)]$; (Initializes the population)
3: $EvalPopulation[P(t)]$; (Evaluates the population)
4: **while** not termination **do**
5: $P'(t) \leftarrow Variation[P(t)]$ (Creation of new individuals)
6: $EvalPopulation[P'(t)]$; (Evaluates the new individuals)
7: $P(t+1) \leftarrow ApplyGeneticOperators[P'(t)]$; (Creation of next generation population)
8: $t \leftarrow t+1$;
9: **end while**

here means, despite of initial condition, the algorithm always could reach to the same or similar solution in the end if ones exists. Under the same problem setting, EA may produces very different results every time Fig. 9. Therefore researchers usually take the average result across different runs to evaluate MOEAs performance, which is not reliable since we could never guarantee its performance every time we run it.

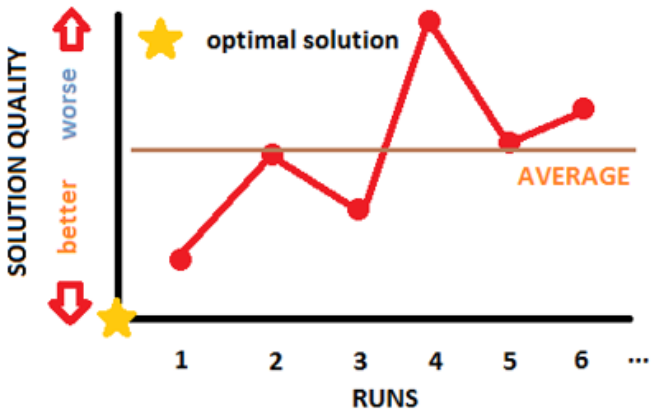


Figure 9: EA is highly unstable, it may produces very different results in each run.

In order to overcome EAs stability issue, we will borrow the stability property from Evolutionary Game Theory which is described in the next section 2.3.

2.3 Game Theory

Game Theory is a study of strategic decision making of conflict and cooperation among intelligent rational decision makers. It is an interactive decision theory which means in a game, given a set of strategies, each player strives to find a strategy that optimizes its own payoff depending on the others strategy decisions. Game theory seeks such strategies for all players as a solution, called Nash equilibrium (NE), where no players can gain extra payoff by unilaterally changing his strategy.

2.3.1 Nash equilibrium

Nash equilibrium is a solution concept in which no player has anything to gain by changing only their own strategy. In a two player game, it is a strategy pair. Let $E(S,T)$ represent the payoff for playing strategy S against strategy T. The strategy pair (S,S) is a Nash equilibrium in a two player game if and only if this is true for both players and for all $T \neq S$.

$$E(S,S) \geq E(T,S)$$

To illustrate the concept of Game Theory and Nash Equilibrium, let's take a look at the classical well known Prisoner's Dilemma Fig. 10. In this game there are two players, prisoner A and B. Each of them has two strategy to chose confess or remain silent. The point is depending on each prisoner's choice they would have different sentences.

1. If A and B both remain silent, then both of them will only serve 1 year in prison.

2. If A confess but B remains silent, then A will be set free and B will serve 20 years in prison (and vice versa).

3. If A and B both confess, then each of them serves 5 years in prison

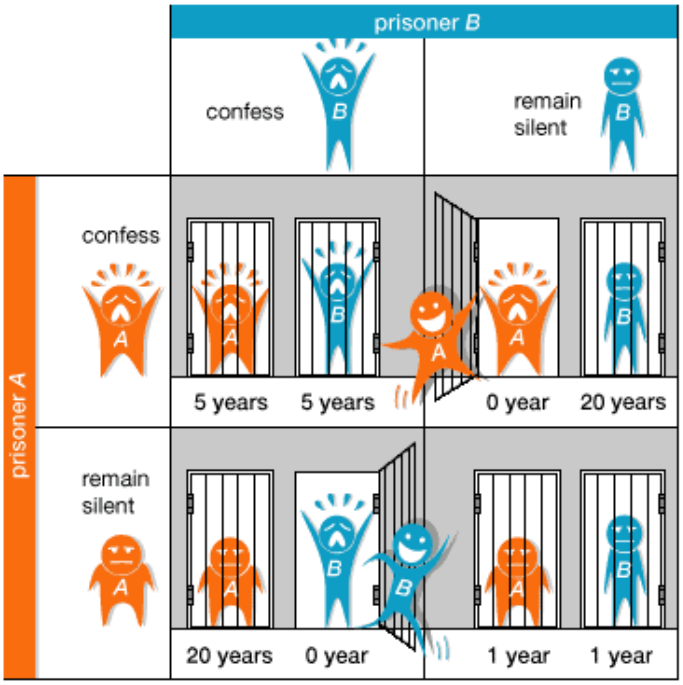


Figure 10: Prisoner's Dilemma

In the case 1, prisoner A or B would go free if they switch their own strategy while another one remains the same. In the case 2, prisoner A or B would serve 15 less years if they switch their own strategy while another one remains the same. Therefore by definition both case 1 and 2 are not Nash equilibrium solution. The only Nash equilibrium in this game is the case 3, where none of prisoners could gain extra payoff by switching their own strategy.

2.4 Evolutionary Game Theory

Evolutionary Game Theory is an application of Game Theory to biological contexts for analyzing population dynamics and stability in biological systems. In EGT, each player maintains a population which is formed by a set of strategies and games are played repeatedly by strategies randomly drawn from the population. In general, EGT considers two major components, Evolutionarily stable strategies (ESS), and Replicator Dynamics (RD).

2.4.1 Evolutionary Stable Strategy

An Evolutionary Stable Strategy (ESS) is anequilibrium refinementof theNash equilibrium. It is a Nash equilibrium that is evolutionarilystable: once it appearsin a population,natural selectionalone is sufficient to prevent alternative (mutant) strategies from invading successfully. ESS specifies two conditions for a strategy S to be an ESS, for all $T \neq S$, either

$$E(S,S) > E(T,S) \text{ or}$$
$$E(S,S) = E(T,S) \text{ and } E(S,T) > E(T,T)$$

The first condition is called a strict Nash equilibrium. The second condition means that although strategy T is neutral with respect to the payoff against strategy S, the population of players who continue to play strategy S has an advantage when playing against T.

Suppose all players in the initial population are programmed to play a certain (incumbent) strategy k. Then, let a small population share of players, $x \in (0,1)$, mutate and play a different (mutant) strategy ℓ. When a player is drawn for a game, the probabilities that its opponent plays k and ℓ are $1-x$ and x, respectively. Thus, the expected payoffs for the player to play k and ℓ are denoted as $U(k, x\ell + (1-x)k)$ and $U(\ell, x\ell + (1-x)k)$, respectively.

Definition 1 *A strategy k is said to be evolutionarily stable if, for every strategy $\ell \neq k$, a certain $\bar{x} \in (0,1)$ exists, such that the inequality*

$$U(k,\ x\ell + (1-x)k) > U(\ell,\ x\ell + (1-x)k) \tag{2.1}$$

holds for all $x \in (0,\bar{x})$.

If the payoff function is linear, Equation 2.1 derives:

$$(1-x)U(k,k) + xU(k,\ell) > (1-x)U(\ell,k) + xU(\ell,\ell) \tag{2.2}$$

If x is close to zero, Equation 2.2 derives either

$$U(k,k) > U(\ell,k) \text{ or } U(k,k) = U(\ell,k) \text{ and } U(k,\ell) > U(\ell,\ell) \tag{2.3}$$

This indicates that a player associated with the strategy k gains a higher payoff than the ones associated with the other strategies. Therefore, no players can benefit by changing their strategies from k to the others. This means that an ESS is a solution on a Nash equilibrium. An ESS is a strategy that cannot be invaded by any alternative (mutant) strategies that have lower population shares.

2.4.2 Replicator Dynamics

The replicator dynamics is a model of evolution that describes how population shares associated with different strategies grows over time [TJ78]. Replicator dynamics assumes infinite population size, continuous infinite time, and complete mixing. Complete mixing means pairwise strategies are completely random chosen from the population.

Let $\lambda_k(t) \geq 0$ be the number of players who play the strategy $k \in K$, where K is the set of available strategies. The total population of players is given by $\lambda(t) = \sum_{k=1}^{|K|} \lambda_k(t)$. Let $x_k(t) = \lambda_k(t)/\lambda(t)$ be the population share of players who play k at time t. The population state is defined by $X(t) = [x_1(t), \cdots, x_k(t), \cdots, x_K(t)]$. Given X, the expected payoff of playing k is denoted by $U(k,X)$. The population's average payoff, which is same as the payoff of a player drawn randomly from the population, is denoted by $U(X,X) = \sum_{k=1}^{|K|} x_k \cdot U(k,X)$.

In the replicator dynamics, the dynamics of the population share x_k is described as follows. $\dot{x}_k$ is the time derivative of x_k.

$$\dot{x}_k = x_k \cdot [U(k,X) - U(X,X)] \tag{2.4}$$

This equation states that players increase (or decrease) their population shares when their payoffs are higher (or lower) than the population's average payoff.

Theorem 2.4.1 *If a strategy k is strictly dominated, then* $x_k(t)_{t\to\infty} \to 0$.

A strategy is said to be strictly dominant if its payoff is strictly higher than any opponents. As its population share grows, it dominates the population over time. Conversely, a strategy is said to be strictly dominated if its payoff is lower than that of a strictly dominant strategy. Thus, strictly dominated strategies disappear in the population over time.

There is a close connection between Nash equilibria and the steady states in the replicator dynamics, in which the population shares do not change over time. Since no players change their strategies on Nash equilibria, every Nash equilibrium is a steady state in the replicator dynamics. As described in Section 2.4.2, an ESS is a solution on a Nash equi-

librium. Thus, an ESS is a solution at a steady state in the replicator dynamics. In other words, an ESS is the strictly dominant strategy in the population on a steady state.

In a conventional game, the objective of a player is to choose a strategy that maximizes its payoff. In contrast, evolutionary games are played repeatedly by all players until a steady point where each player finds their strictly dominant strategy. The strictly dominant strategy in the population is an ESS which is a solution on a Nash Equilibrium that is evolutionarily stable. The Evolutionary game model is illustrated in Fig. 11 and described as follow.

1. The evolution model deals with a Population (G) at generation n. Competition happens among strategies within the population and it is represented by the Game.

2. N/2 Games are performed in each generation. Each Game tests the strategies in pairwise under the rules of the game. These rules produce different objective payoffs.

3. Based on the payoff the winner replaces the loser in each game.

4. This overall process then produces a new Population (G+1). And the new population then takes the place of the previous one and the cycle begins again (and never stops).

5. It is aniterative process, over time only one strictly dominant strategy will stands in the population, which cannot be invaded by any new mutant strategies. And it is by definition an Evolutionary Stable Strategy (ESS).

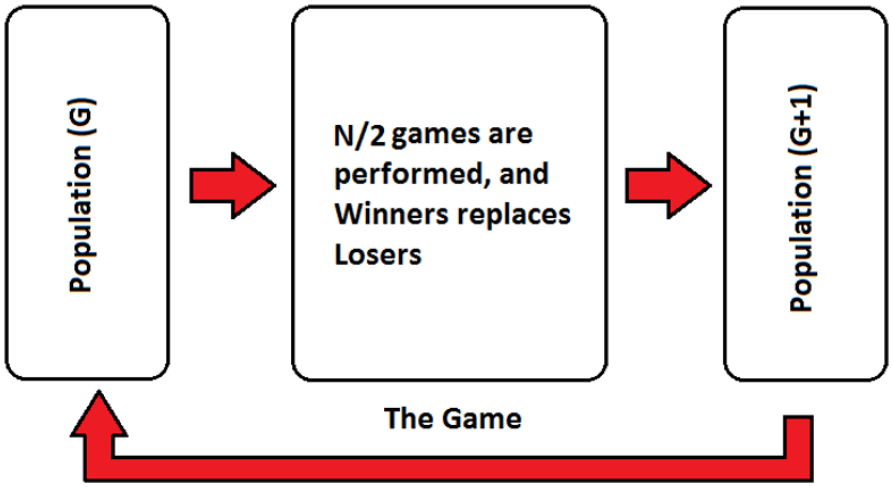


Figure 11: Evolution process in Evolutionary Game Theory

CHAPTER 3

EVOLUTIONARY GAME THEORETIC MULTI-OBJECTIVE ALGORITHMS (EGTMOA)

Evolutionary Game Theoretic Multi-Objective Algorithms is an Evolutionary Multi-Objective Algorithm designed to follow Evolutionary Game Theory scheme with the goal to seek for a global optimal evolutionarily stable solution. EGTMOA combines the stability property from Evolutionary Game Theory and optimality notion from Evolutionary Algorithms together to form a new metaheuristic multi-objective algorithm that seeks for a set of strict dominant strategies as a global optimal and stable solution through evolution mechanism in a reasonable running time. It has follow properties.

- **Optimality**: Seeking for a set of strict dominant strategies as a global optimal solution.
- **Stability**: Providing a stable solution by minimizing oscillations in decision makings.
- **Metaheuristic**: Just like MOEAs, EGTMOA does not make any assumption about the solution space, and it does not require any domain knowledge about the problem to be solved.

EGTMOA does not guarantee to find a true optimal solution, but instead it provides a high quality stable approximation solution to multi-objective problems at the end.

3.1 Baseline Algorithm

EGTMOA is an evolutionary algorithm in which the payoff of a strategy is evaluated based on their interactions with other strategies. EGTMOA divides the entire multi-objective problem into M sub problems Fig. 12. Each sub problem is handled by an agent or player that maintains a population of N strategies. And each player seeks a strictly dominant strategy that maximize its payoff interacting with other players through generations.

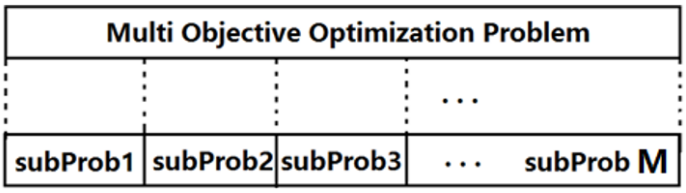


Figure 12: Dividing the entire whole multi-objective problem into M sub problems.

A baseline EGTMOA algorithm is described in Algorithm 3.1.1 and the evolution of a population is illustrated in Fig. 13. EGTMOA has following main steps.

1. Initially random generate N strategies (decision variables) for each population (Line 2).
2. Random shuffle the order of populations to compute (Line 6).
3. For each population perform N/2 pairwise game evaluation (Line 6-16). A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (performGame() in Algorithm 3.1.2).
4. Winner duplicates itself, loser is deleted from the population (Line 10,14,15).
5. The duplicated winner has a probability to be mutated (Line 11-13).
6. Check the termination. If it satisfies then go to Step 8, otherwise back to Step 2 (next generation).

7. Take the feasible strategy with the greatest population share as dominant strategy d_i (Line 18-22).

8. Return the dominant strategy d_i of each population as a global solution for the formulated Multi-Objective Optimization Problem (Line 26).

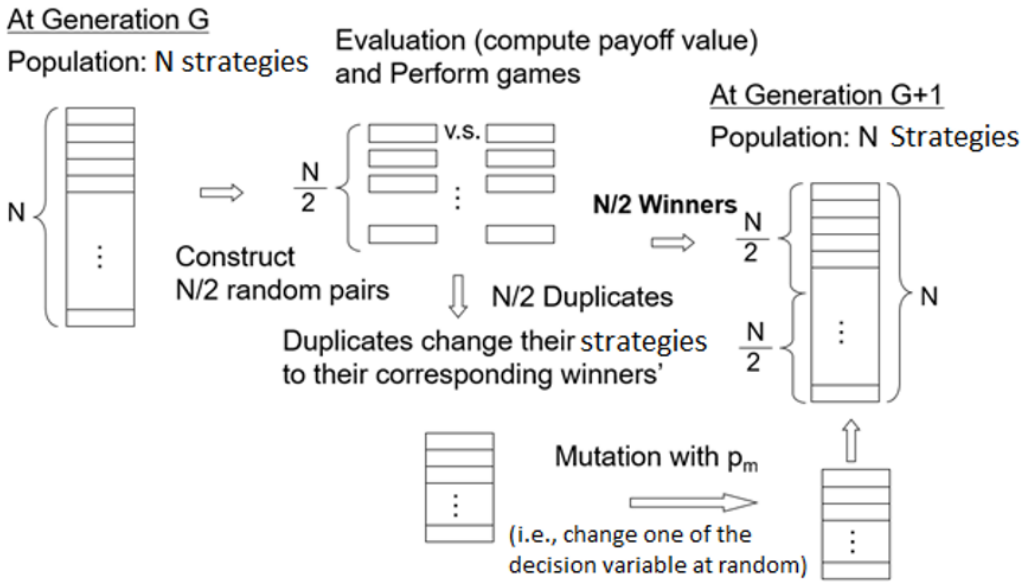


Figure 13: Evolution workflow of a population in Baseline EGTMOA algorithm.

3.2 Quality Indicators

From each population we randomly chose $N/2$ pairwise strategies to play the game Algorithm 3.1.2. In each game a winner and a loser will be determined based on their solution quality. Thus, a game is performed by comparing the fitness values of the chosen pair strategies. These fitness values are computed through objective functions that are defined by multi-objective problems formulation. Since we are dealing with two or more objectives we need a mechanism that helps us to determine the winner and loser in a multi-objective

point of view. And this mechanism is handled by quality indicator, for the baseline EGT-MOA algorithm we use the Pareto Dominance (PD) notion as our primary quality indicator [SD95]..

3.2.1 Pareto Dominance

Pareto Dominance guarantees the optimality of one strategy over another. It is often called strict dominance quality indicator, because a strategy must have equal or better fitness value in all the objectives in order to become the winner. A strategy s_1 is said to dominate another strategy s_2 if both of the following conditions hold:

- s_1's objective values are superior than, or equal to, s_2's in all objectives.
- s_1's objective values are superior than s_2's in at least one objectives.

The dominating strategy wins a game over the dominated one. If two strategies are non-dominated with each other, the winner is randomly selected. To illustrate better the concept of Pareto Dominance, let's take a look at Fig. 14. Considering a two objective minimization problem we try to find the best solution among $s1, s2$ and $s3$ using Pareto Dominance notion. From the Fig. 14 we can easily conclude that $s2$ performs better than $s3$ in one objective and worse in another one. Therefore $s2$ and $s3$ are non-dominated with each other. And it is clear that $s1$ performs better in all two objectives than other two solutions. So we say $s1$ dominates both $s2$ and $s3$. Thus, $s1$ is the winner after playing game against $s2$ and $s3$.

3.2.2 Hypervolume

If the number of conflicting objectives is more than 2, then most of the time strategies are non-dominated with each other which make the population hard to evolve toward an ESS.

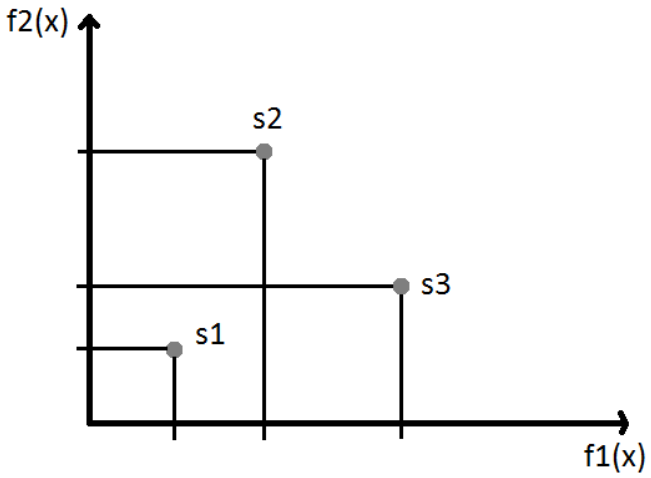


Figure 14: Quality comparison with two objectives using Pareto Dominance.

In this case we employ a Hypervolume metric [ZT98] based quality indicator. It measures the volume that a given strategy (s) dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{3.1}$$

Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. i.e. in Fig. 15 we have three solution points in the objective space P_1, P_2 and P_3. R represents the reference point, it usually takes the maximum value that each objective function could reach. Now we compute each solution's Hypervolume value as the rectangle area that is formed by each of the solution point with the reference point. A higher Hypervolume value means that a solution is more optimal. Thus, P_2 is the winner after playing game against P_2 and P_3. It should be noted that before we compute the Hypervolume value, we have to normalized all the objectives into interval $[0, 1]$.

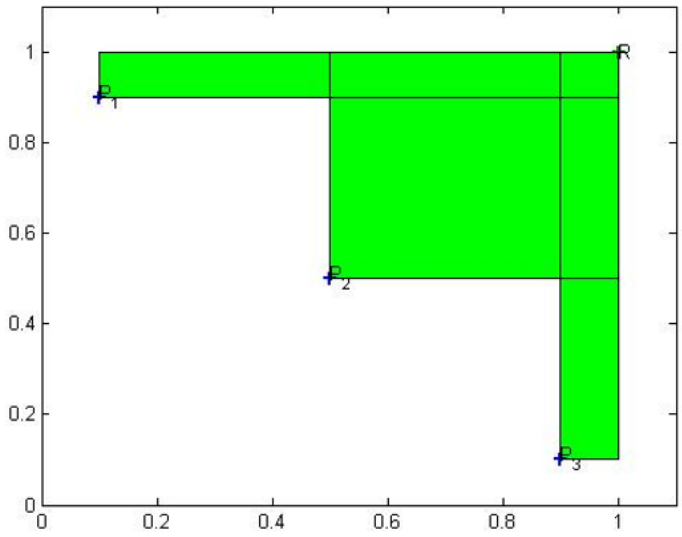


Figure 15: Quality comparison with two objectives using Hypervolume.

3.3 Constraints handling

Multi-objective optimization problems often present many constraints which make the problem more realistic, but harder to be solved. Those constraints are defined by a set of equality and inequality functions. EGTMOAs take constraints into account every time a game is performed Algorithm 3.1.2.

If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins the game. If both strategies are feasible, they are compared using a quality indicator.

If both strategies are infeasible in a game, then they are compared based on their constraint violation. Constraint violation is computed as the difference between the constraint value and its cap limit value. An infeasible strategy s_1 wins a game over another infeasible strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

3.4 Mutation

After each game the winner duplicates itself and loser is deleted from the population. Due to this dominance notion EGTMOA some time may converges very fast into a local optimal point. In order to prevent the mentioned problem, we need to keep a certain diversity in the population that helps us to move toward a global optimal solution. In this case mutation does the job, it helps us to jump out from a local optimal point. Mutation is a biological operator that is applied with a certain probability to the winner of each game shown in Algorithm 3.1.1 (Line 11-13). Te baseline EGTMOA algorithm uses Polynomial Mutation which is described in the next Section 3.4.1.

3.4.1 Polynomial Mutation

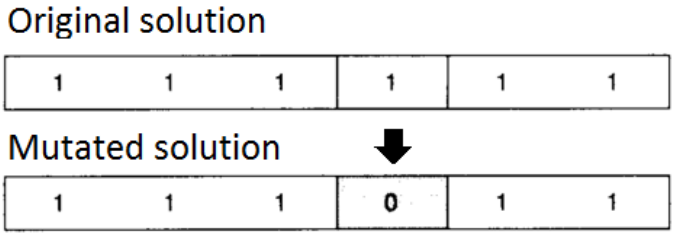


Figure 16: An example of polynomial mutation with 6 decision variables.

In polynomial mutation each decision variable has a probability of $1/v$ to be mutated, where v is the total number of decision variables. The mutated value is randomly generated

within the same range as the original decision variable value. i.e. in Fig. 16 $v = 6$, so each decision variable mutates with a probability of $1/6$.

3.5 Termination

EGTMOA performs iterative evolution process through generations seeking for the set of strictly dominant strategy as a global optimal solution. In theory the evolution process assumes infinite population size, and continuous infinite time. However in practice this is not feasible, we have to stop the algorithm in some point. Therefore we need to define a termination criteria that tells us when we should stop running our EGTMOA and to get the final solution result. EGTMOA provides two different approaches for defining the termination condition.

- **Static termination**: Chose a fixed maximum number of generations G based on experiments and empirical results.
- **Dynamic termination**: EGTMOA ends if the dominant strategy does not improve more than $X\%$ of performance within Y number of generations. Or the dominant strategy has a population share greater than $Z\%$. It should be noted that the value of X, Y and Z come from experiments and empirical results.

Thanks to EGTMOA's stability property, its convergence curve, speed and performance are stable as well across different runs which makes the choice of proper X, Y, Z and G values more reliable. In the next Section 3.6 we are going to see a stability analysis of the proposed algorithm EGTMOA.

3.6 Stability Analysis

This section analyzes EGTMOA's stability (i.e., reachability to at least one of Nash equilibria) by proving the state of each population converges to an evolutionarily stable equilibrium. The proof consists of three steps: (1) designing a set of differential equations that describe the dynamics of the population state (or strategy distribution), (2) proving an strategy selection process has equilibria and (3) proving the the equilibria are asymptotically stable (or evolutionarily stable) . The proof uses the following terms and variables.

- S denotes the set of available strategies. S^* denotes a set of strategies that appear in the population.
- $X(t) = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}$ denotes a population state at time t where $x_s(t)$ is the population share of a strategy $s \in S$. $\sum_{s \in S^*}(x_s) = 1$.
- F_s is the fitness of a strategy s. It is a relative value determined in a game against an opponent based on the dominance relationship between them. The winner of a game earns a higher fitness than the loser.
- $p_k^s = x_k \cdot \phi(F_s - F_k)$ denotes the probability that a strategy s is replicated by winning a game against another strategy k. $\phi(F_s - F_k)$ is the probability that the fitness of s is higher than that of k.

The dynamics of the population share of s is described as follows.

$$\dot{x_s} = \sum_{k \in S^*, k \neq s} \{x_s p_k^s - x_k p_s^k\} = x_s \sum_{k \in S^*, k \neq s} x_k \{\phi(F_s - F_k) - \phi(F_k - F_s)\} \tag{3.2}$$

Note that if s is strictly dominated, $x_s(t)_{t \to \infty} \to 0$.

Theorem 3.6.1 *The state of a population converges to an equilibrium.*

Proof It is true that different strategies have different fitness values. In other words, only one strategy has the highest fitness among others. Given Theorem 2.4.1, assuming that $F_1 > F_2 > \cdots > F_{|S^*|}$, the population state converges to an equilibrium: $X(t)_{t\to\infty} = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}_{t\to\infty} = \{1, 0, \cdots, 0\}$.

Theorem 3.6.2 *The equilibrium found in Theorem 3.6.1 is asymptotically stable.*

Proof At the equilibrium $X = \{1, 0, \cdots, 0\}$, a set of differential equations can be downsized by substituting $x_1 = 1 - x_2 - \cdots - x_{|S^*|}$

$$\dot{z}_s = z_s[c_{s1}(1 - z_s) + \sum_{k=2, k\neq s}^{|s^*|} z_k \cdot c_{sk}], \quad s, k = 2, ..., |S^*| \tag{3.3}$$

where $c_{sk} \equiv \phi(F_s - F_k) - \phi(F_k - F_s))$ and $Z(t) = \{z_2(t), z_3(t), \cdots, z_{|S^*|}(t)\}$ denotes the corresponding downsized population state. Given Theorem 2.4.1, $Z_{t\to\infty} = Z_{eq} = \{0, 0, \cdots, 0\}$ of $(|S^*| - 1)$-dimension.

If all Eigenvalues of Jaccobian matrix of $Z(t)$ has negative real parts, Z_{eq} is asymptotically stable. The Jaccobian matrix J's elements are

$$\begin{aligned} J_{sk} &= \left[\frac{\partial \dot{z}_s}{\partial z_k}\right]_{|Z=Z_{eq}} = \left[\frac{\partial z_s[c_{s1}(1 - z_s) + \sum_{k=2, k\neq s}^{|S^*|} z_k \cdot c_{sk}]}{\partial z_k}\right]_{|Z=Z_{eq}} \\ &\text{for } s, k = 2, ..., |S^*| \end{aligned} \tag{3.4}$$

Therefore, J is given as follows, where $c_{21}, c_{31}, \cdots, c_{|S^*|1}$ are J's Eigenvalues.

$$J = \begin{bmatrix} c_{21} & 0 & \cdots & 0 \\ 0 & c_{31} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{|S^*|1} \end{bmatrix} \tag{3.5}$$

$c_{s1} = -\phi(F_1 - F_s) < 0$ for all s; therefore, $Z_{eq} = \{0, 0, \cdots, 0\}$ is asymptotically stable.

Algorithm 3.1.1: Evolutionary Game Theoretic Multi-Objective Algorithm

1: $g = 0$
2: Randomly generate the initial N populations: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$
3: **while** not termination **do**
4: **for** each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
5: $\mathscr{P}'_i \leftarrow \emptyset$
6: **for** $j = 1$ to $|\mathscr{P}_i|/2$ **do**
7: $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
8: $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
9: $winner \leftarrow$ performGame(s_1, s_2)
10: $replica \leftarrow$ replicate($winner$)
11: **if** random() $\leq P_m$ **then**
12: $replica \leftarrow$ mutate($winner$)
13: **end if**
14: $\mathscr{P}_i \setminus \{s_1, s_2\}$
15: $\mathscr{P}'_i \cup \{winner, replica\}$
16: **end for**
17: $\mathscr{P}_i \leftarrow \mathscr{P}'_i$
18: $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
19: **while** d_i is infeasible **do**
20: $\mathscr{P}_i \setminus \{d_i\}$
21: $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
22: **end while**
23: **end for**
24: $g = g + 1$
25: **end while**
26: return $d = \{d_1, d_2, ..., d_N\}$

Algorithm 3.1.2: Game between Strategies - performGame()

Require: s_1 and s_2: Strategies to play a game

Ensure: Winner of the game

1: **if** s_1 and s_2 are feasible **then**
2: **if** $s_1 \succ s_2$ **then**
3: **return** s_1
4: **end if**
5: **if** $s_2 \succ s_1$ **then**
6: **return** s_2
7: **end if**
8: **return** randomlySelect($\{s_1, s_2\}$)
9: **end if**
10: **if** s_1 is feasible and s_2 is infeasible **then**
11: **return** s_1
12: **end if**
13: **if** s_2 is feasible and s_1 is infeasible **then**
14: **return** s_2
15: **end if**
16: **if** s_1 and s_2 are infeasible **then**
17: **return** $argmin_{s \in \{s_1, s_2\}} c_s^v$
18: **end if**

CHAPTER 4

VIRTUAL MACHINE DEPLOYMENT ON CLOUD DATA CENTER

4.1 Introduction

It is a challenging issue for cloud operators to place applications so that the applications can satisfy given constraints in performance (e.g. response time) while maintaining their resource utilization (CPU and network bandwidth utilization) and power consumption. The operators are required to dynamically place applications by adjusting their locations and resource allocation according to various operational conditions such as workload and resource availability. In order to address this challenge, this Chapter investigates three different application placement schedulers, Cielo, AGEGT, and Cielo-LP which exhibit the following properties:

- **Self-optimization**: allows applications to autonomously seek their optimal placement configurations (i.e., locations and resource allocation) according to operational conditions (e.g., workload and resource availability), as adaptation decisions, under given optimization objectives and constraints.
- **Self-stabilization**: allows applications to autonomously seek stable adaptation decisions by minimizing oscillations (or non-deterministic inconsistencies) in decision making.

Cielo, AGEGT, and Cielo-LP are EGTMOA framework based algorithms that approach the self-optimization and self-stabilization properties with Evolutionary Algorithm (EA) and Evolutionary Game Theory (EGT), respectively. In general, EGTMOAs are robust problem-independent search methods that seek optimal solutions (adaptation decisions) with reasonable computational costs by maintaining a small ratio of search coverage to the entire search space [Eib02, Deb01]. EGTMOAs employ EGT as a means to mathematically formulate adaptive decision making and theoretically guarantee that each decision making process converges to an evolutionarily (or asymptotically) stable equilibrium where a specific (stable) adaptation decision is deterministically made under a particular set of operational conditions. EGTMOAs allow applications to seek the solutions to optimally adapt their locations and resource allocation and operate at equilibria by making evolutionarily stable decisions for application placement.

4.2 State of the art

Numerous research efforts have been made to study heuristic algorithms for application placement problems in clouds (e.g., [LQL13, MLL12, GP13, CSV12, LWY09, KBK13, CJH10, WKL10]). Most of them assume a single-tier application architecture and considers a single optimization objective. For example, in [LWY09, KBK13, CJH10, WKL10], only power consumption is considered as the objective. In contrast, I assume a multi-tier application architecture (i.e., three tiers in an application) and considers multiple objectives. It is designed to seek a trade-off solution among conflicting objectives.

Game theoretic algorithms have been used for a few aspects of cloud computing; for example, application placement [KA09b, WVZ09, DDL07], task allocation [SZL08] and data replication [KA09a]. In [KA09b, WVZ09, DDL07], greedy algorithms seek equilibria

in application placement problems. This means they do not attain the stability property to reach equilibria as EGTMOAs do.

Several genetic algorithms (e.g., [WSY12, TEC08]) and other stochastic optimization algorithms (e.g., [GGQ13, CWJ13]) have been studied to solve application placement problems in clouds. They seek the optimal placement solutions; however, they do not consider stability. In contrast, EGTMOAs aid applications to seek evolutionarily stable solutions and stay at equilibria.

This study is novel in that EGTMOAs integrate optimization and stabilization processes to seek optimal and stable solutions. Optimization and stabilization have been studied largely in isolation, but few attempts have been made so far to integrate and facilitate them simultaneously, except in a very limited number of work (e.g., [KL03]).

Evolutionary algorithms and other stochastic search algorithms often focus on optimization and fail to seek stable solutions [MF04, Kun99, MW96]. As a result, they can inconsistently yield different sets of solutions in different runs/trials with the same problem settings, especially when a given problem's search space is large [TD08, YKK08, LZW08, LP07]. Conversely, EGTMOAs are often dedicated to seek stable solutions (i.e., equilibria), which are not necessarily optimal [Wei96, Now06, NRT07].

4.3 Problem Statement

This section formulates an application placement problem to place N applications on M hosts available in a cloud data center. Each application is designed with a set of server software, following a three-tier application architecture [UPS05, SH06] (Fig. 17).

Using a certain hypervisor such as Xen [BDF03], each server is assumed to run on a virtual machine (VM) atop a host. A host can operate multiple VMs. They share resources

available on their local host. Each host is assumed to be equipped with a multi-core CPU that supports DVFS in each core.

Each message is sequentially processed from a Web server to a database server through an application server. A reply message is generated by the database server and forwarded in the reverse order toward a user. (Fig. 17). It is assumed that different applications utilize different sets of servers. (Servers are not shared by different applications.) And each host runs multi cores processor to allocate different applications.

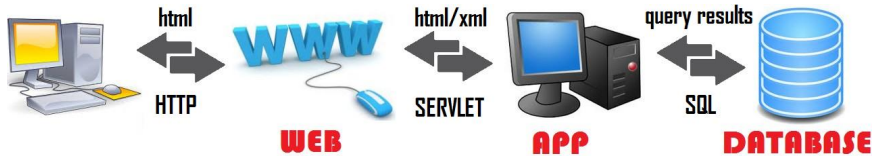


Figure 17: Three-Tiered Application Architecture

The goal of this problem is to find evolutionarily stable strategies that deploy N applications (i.e., $N \times 3$ VMs) on M hosts so that the applications adapt their locations and resource allocation to given workload and resource availability with respect to the four objectives described below. Every objective is computed on an application by application basis and is to be minimized.

- **CPU allocation** (f_C): A certain CPU time share (in percentage) is allocated to each VM. (The CPU share of 100% means that a CPU core is fully allocated to a VM.) It represents the upper limit for the VM's CPU utilization. This objective is computed as the sum of CPU shares allocated to three VMs of an application.

$$f_C = \sum_{t=1}^{3} c_t \tag{4.1}$$

  c_t denotes the CPU time share allocated to the t-th tier server in an application.

- **Bandwidth allocation** (f_B): A certain amount of bandwidth (in bits/second) is allocated to each VM. It represents the upper limit for the VM's bandwidth consumption. This objective is computed as the sum of bandwidth allocated to three VMs of an application.

$$f_B = \sum_{t=1}^{3} b_t \tag{4.2}$$

b_t denotes the amount of bandwidth allocated to the t-th tier server in an application.

- **Response time** (f_{RT}): This objective indicates the time required for a message to travel from a web server to a database server:

$$f_{RT} = T^p + T^w + T^c \tag{4.3}$$

T^p denotes the total time for an application to process an incoming message from a user at three servers. T^w is the waiting time for a message to be processed at servers. T^c denotes the total communication delay to transmit a message between servers. T^p, T^w and T^c are estimated with the $M/M/M$ queuing model, in which message arrivals follow a Poisson process and a server's message processing time is exponentially distributed.

T^p is computed as follows where T_t^p denotes the time required for the t-th tier server to process a message.

$$T^p = \sum_{t=1}^{3} T_t^p \tag{4.4}$$

T^w is computed as follows.

$$T^w = \frac{1}{\lambda}\sum_{t=1}^{3}\rho_0\frac{a_t^O}{O!}\frac{\rho_t}{(1-\rho_t)^2} \tag{4.5}$$

$$\text{where } a_t = \lambda_t\frac{T_t^p}{c_t\cdot q_t/q_{max}},\ \rho_t = \frac{a_t}{O},\ \rho_0 = \left(\sum_{n=0}^{O-1}\frac{\rho_t^n}{n!} + \frac{\rho_t^O}{O!}\frac{1}{1-\rho_t/O}\right)^{-1}$$

λ denotes the message arrival rate for an application (i.e., the number of messages the application receives from users in the unit time). Note that $\lambda = \frac{1}{3}\sum_{t=1}^{3}\lambda_t$ where λ_t is the message arrival rate for the t-th tier server in the application. Currently, $\lambda = \lambda_1 = \lambda_2 = \lambda_3$. ρ_t denotes the utilization of a CPU core that the t-th tier server resides on. q_{max} is is the maximum CPU frequency. q_t is the frequency of a CPU core that the t-th tier server resides on. O is the total number of cores that a CPU contains.

T^c is computed as follows.

$$T^c = \sum_{t=1}^{2}T_{t\to t+1}^c \approx \sum_{t'=2}^{3}\frac{B\cdot\lambda_{t+1}}{b_t} \tag{4.6}$$

B is the size of a message (in bits). $T_{t\to t+1}^c$ denotes the communication delay to transmit a message from the t-th to $(t+1)$-th server. b_t denotes the bandwidth allocated to the t-th tier server (bits/second).

- **Power Consumption** (f_{PC}): This objective indicates the total power consumption (in Watts) by the CPU cores that operate three VMs in an application.

$$f_{PC} = \sum_{t=1}^{3}\left(P_{idle}^{q_t} + (P_{max}^{q_t} - P_{idle}^{q_t})\cdot c_t\cdot\frac{q_t}{q_{max}}\right) \tag{4.7}$$

$P_{idle}^{q_t}$ and $P_{max}^{q_t}$ denote the power consumption of a CPU core that the t-th tier server resides on when its CPU utilization is 0% and 100% at the frequency of q_t, respectively.

Four constraints are considered.

- **CPU core capacity constraint** (C_C): The upper limit of the total share allocation on each CPU core. $c_{i,o} \leq C_C$ for all O cores on all M hosts where $c_{i,o}$ is the total share allocation on the o-th core of the i-th host. The violation of this constraint is computed as:

$$g_C = \sum_{i=1}^{M} \sum_{o=1}^{O} \left(I_{i,o}^{C} \cdot (c_{i,o} - C_C) \right) \tag{4.8}$$

  $I_{i,o}^{C} = 1$ if $o_i > C_C$. Otherwise, $I_{i,o}^{C} = 0$.

- **Bandwidth capacity constraint** (C_B): The upper limit of bandwidth consumption allocated to each host. $b_i \leq C_B$ for all M hosts where b_i is the total amount of bandwidth allocated to the i-th host. The violation of this constraint is computed as:

$$g_B = \sum_{i=1}^{M} \left(I_i^B \cdot (b_i - C_B) \right) \tag{4.9}$$

  $I_i^B = 1$ if $b_i > C_B$. Otherwise, $I_i^B = 0$.

- **Response time constraint** (C_{RT}): The upper limit of response time for each application. $f_{RT}^{i} \leq C_{RT}$ for all applications where f_{RT}^{i} is the response time of the i-th application. The violation of this constraint is computed as:

$$g_{RT} = \sum_{i=1}^{N} \left(I_i^{RT} \cdot (f_{RT}^{i} - C_{RT}) \right) \tag{4.10}$$

  $I_i^{RT} = 1$ if $f_{RT}^{i} > C_{RT}$. Otherwise, $I_i^{RT} = 0$.

- **Power consumption constraint** (C_{PC}): The upper limit of power consumption for each application. $f_{PC}^{i} \leq C_{PC}$ for all N applications where f_{PC}^{i} is the power consumption of the i-th application. The violation of power consumption constraint is computed as:

$$g_{PC} = \sum_{i=1}^{N} \left(I_i^{PC} \cdot (f_{PC}^{i} - C_{PC}) \right) \tag{4.11}$$

$I_i^{PC} = 1$ if $f_{PC}^{i} > C_{PC}$. Otherwise, $I_i^{PC} = 0$.

In this Chapter three different variants of EGTMOA based framework are studied, Cielo, AGEGT, and Cielo-LP. Each of them occupies a Section in this Chapter, and it is organized starting with a brief introduction, following by the algorithm description and it ends with experiment results and conclusions.

4.4 Cielo

This section studies a EGTMOA framework for application placement in clouds that support a power capping mechanism (e.g., Intel's Runtime Average Power Limit–RAPL) for CPUs. Given the notion of power capping, power can be treated as a schedulable resource in addition to traditional resources such as CPU time share and bandwidth share. The proposed algorithm is called Cielo (Sky in Spanish), aids cloud operators to schedule resources (e.g., power, CPU and bandwidth) to applications and place applications onto particular CPU cores in an adaptive and stable manner according to the operational conditions in a cloud, such as workload and resource availability. This study evaluates Cielo through a theoretical analysis and simulations. It is theoretically guaranteed that Cielo allows each application to perform an evolutionarily stable deployment strategy, which is an equilibrium solution under given operational conditions. Simulation results demonstrate that Cielo

allows applications to successfully leverage the notion of power capping to balance their response time performance, resource utilization and power consumption.

4.4.1 Introduction

Dynamic Voltage and Frequency Scaling (DVFS) is a major method of choice for investigating the trade-off between power consumption and performance in cloud applications. Power capping is an emerging alternative to DVFS [RAS]. Instead of managing the CPU's frequency directly, the user simply specifies a time window and a power consumption bound. The CPU guarantees that its average power consumption will not exceed the specified could over each window. Both the window size and bound can be modified at runtime. This mechanism treats power as a schedulable resource and allows cloud operators to control the exact amount of power that each CPU consumes.

Given the current availability of power capping mechanisms from major CPU manufacturers, such as Intel's Runtime Average Power Limit (RAPL), Cielo focuses on an application placement problem for cloud operators to schedule resources (e.g., power, CPU and bandwidth) to applications and place applications onto particular CPU cores according to the operational conditions in a cloud, such as workload and resource availability.

Cielo is a variant of EGTMOA framework for adaptive and stable application placement in clouds that support a power capping mechanism for CPUs. This section describes its design and evaluates its optimality and stability. In Cielo, each application maintains a set (or a population) of deployment strategies, each of which indicates the location of and resource allocation for that application. Cielo theoretically guarantees that, through a series of evolutionary games between deployment strategies, the population state (i.e., the distribution of strategies) converges to an evolutionarily stable equilibrium, which is always converged to regardless of the initial state. (A dominant strategy in the evolutionarily

stable population state is called an **evolutionarily stable strategy**.) In this state, no other strategies except an evolutionarily stable strategy can dominate the population. Given this theoretical property, Cielo aids each application to operate at equilibria by using an evolutionarily stable strategy for application deployment in a deterministic (i.e., stable) manner.

Simulation results verify this theoretical analysis. Applications seek equilibria to perform evolutionarily stable deployment strategies and adapt their locations and resource allocations to given operational conditions. Cielo allows applications to successfully leverage the notion of power capping and balance their response time performance, resource utilization and power consumption. In comparison to existing heuristics, Cielo outperforms two well-known heuristics algorithm first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [LQL13, MLL12, GP13, CSV12].

4.4.2 Algorithm

Cielo maintains N populations, $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$, for N applications and performs games among strategies in each population. A strategy s is defined to indicate the locations of and resource allocation for three VMs in an application:

$$s(a_i) = \bigcup_{t \in 1,2,3} (h_{i,t}, c_{i,t}, u_{i,t}, b_{i,t}, p_{i,t}), \;\; 1 < i < N \tag{4.12}$$

a_i denotes the i-th application. $h_{i,t}$ is the ID of a host that operates a_i's t-th tier VM. $c_{i,t}$ is the ID of the core inside the host $h_{i,t}$. $u_{i,t}$ and $b_{i,t}$ are the CPU and bandwidth allocation for a_i's t-th tier VM. $p_{i,t}$ denotes the power cap of host $h_{i,t}$ core $c_{i,t}$ where allocates t-th

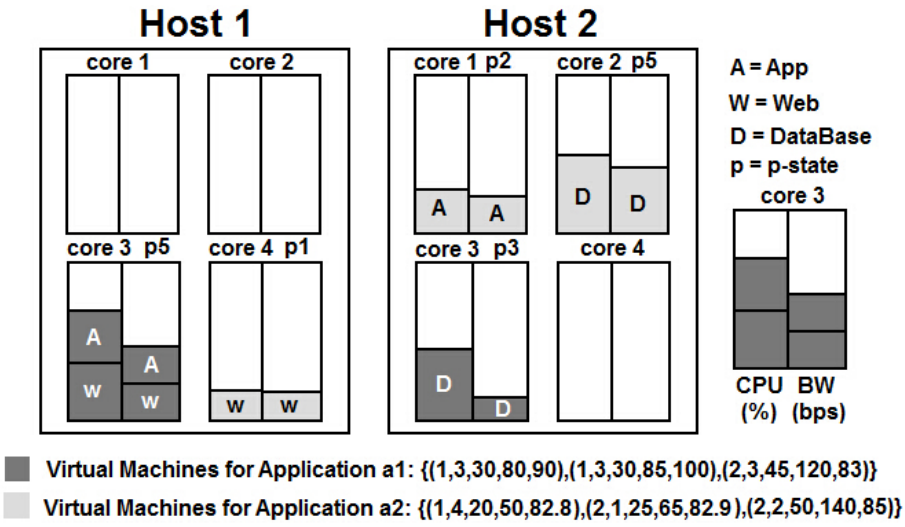


Figure 18: Example of Cielo Deployment Strategies

tier VM. This power cap is translated later to CPU p-state based on the table 3. Each core operates at the highest p-state required by its allocated VMs.

Fig. 18 shows two example strategies for two applications (a_1 and a_2) ($N = 2$ and $M = 2$). a_1's strategy ($s(a_1)$) places the first-tier VM on host 1 core 3($h_{1,1} = 1$, $c_{1,1} = 3$), which caps power to 90 Watts $p_{1,1} = 90$and consumes 30% CPU share and 80 Kbps bandwidth for the VM ($c_{1,1} = 30$ and $b_{1,1} = 80$). The second-tier VM is placed on host 1 core 3($h_{1,2} = 1$, $c_{1,2} = 3$), which caps power to 100 Watts ($p_{1,2} = 100$) and consumes 30% CPU share and 85 Kbps bandwidth for the VM ($c_{1,2} = 30$ and $b_{1,2} = 85$). The third-tier VM is placed on host 2 core 3 ($h_{1,3} = 2$, $c_{1,3} = 3$), which caps power to 83 Watts $p_{1,3} = 83$ and consumes 45% CPU share and 120 Kbps bandwidth for the VM ($c_{1,3} = 45$ and $b_{1,3} = 120$). Given $s(a_1)$, a_1's objective values for CPU allocation and bandwidth allocation are 105% (30 + 30 + 45) and 285 kbps (80 + 85 + 120).

Algorithm 4.4.1 shows how Cielo seeks an evolutionarily stable strategy for each application through evolutionary games. In the 0-th generation, strategies are randomly gener-

ated for each population (Line 2). In each generation (g), a series of games are carried out on every population (Lines 4 to 24). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to the objectives described in Section 4.3 (Lines 7 to 9). The loser disappears in the population. The winner is replicated to increase its population share and mutated with a certain rate P_m (Lines 10 to 15). Mutation randomly chooses one of three VMs in the winner and alters its $h_{i,t}$, $c_{i,t}$ and $b_{i,t}$ values at random (Line 12).

Once all strategies have played games in the population, Cielo identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 18 to 22). A strategy is said to be feasible if it never violate the CPU and bandwidth capacity constraints ($c^v = 0$ in Eq. 4.8 and $b^v = 0$ in Eq. 4.9). It is said to be infeasible if $c^v > 0$ or $b^v > 0$. Cielo deploys three VMs for an application in question based on the dominant strategy.

A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (performGame() in Algorithm 4.4.1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins over its opponent. If both strategies are feasible, they are compared with one of the following three schemes to select the winner.

- Pareto dominance (PD): This scheme is based on the notion of dominance citesrinivas95multiobjective, in which a strategy s_1 is said to dominate another strategy s_2 (denoted by $s_1 \succ s_2$) if both of the following conditions hold:
  - s_1's objective values are superior than, or equal to, s_2's in all objectives.
  - s_1's objective values are superior than s_2's in at least one objectives.

The dominating strategy wins a game over the dominated one. If two strategies are non-dominated with each other, the winner is randomly selected.

- Hypervolume (HV): This scheme is based on the hypervolume metric [ZT98]. It measures the volume that a given strategy (s) dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{4.13}$$

  Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

- Hybrid of Pareto comparison and hypervolume (PD-HV): This scheme is a combination of the above two schemes. First, it performs the Pareto dominance (PD) comparison for given two strategies. If they are non-dominated, the hypervolume (HV) comparison is used to select the winner. If they still tie with the hypervolume metric, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. A strategy s_1 wins a game over another strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

4.4.3 Experiment

This section evaluates Cielo through simulations and discusses how Cielo allows applications to adjust their performance through evolutionary games.

4.4.3.1 Experiment Setting

It uses a simulated cloud data center that consists of 100 hosts in a 10×10 grid topology ($M = 100$). The grid topology is chosen based on recent findings on efficient topology configurations in clouds [GWT08, GLL09]. It also assumes five types of applications. Table 1 shows the message arrival rate (the number of incoming messages per second) and message processing time (second) for each of the five application types. This configuration follows Zipf's law. The experiment simulates 40 application instances for each type (200 application instances in total; $N = 200$).

In this experiment we assume each host is equipped with an Intel Core2 Quad Q6700 CPU, which has five frequency and voltage operating points (P-states). Table 3 shows the power consumption at each P-state under the 0% and 100% CPU utilization [GMC13]. This setting is used in Eq. 4.7 to compute power consumption objective values.

In Cielo, the number of strategies is 100 in each population. Polynomial mutation (P_m in Algorithm 4.4.1) with distribution index 45. The maximum number of generations (G_{max} in Algorithm 4.4.1) is set to 500. Every simulation result is the average with 20 independent simulation runs.

4.4.3.2 Experiment Results

Figs. 19 to 21 illustrate how Cielo three variants evolve deployment strategies through generations and improve their objective values.

Figs. 19(a), 20(a) and 21(a) show that CPU allocation decreases through generations. Cielo HV reaches 8.04% of average in the last generation, which is the best performance among all three Cielo variants.

Figs. 19(b), 20(b) and 21(b) show that BW allocation improves over generations. Cielo HV reaches 200.95 bps of average in the last generation, which is the best performance among all three Cielo variants.

Figs. 19(c), 20(c) and 21(c) show that Cielo successfully saves energy consumption through generations. Cielo HV reaches 334.89 Watts of average in the last generation, which is the best performance among all three Cielo variants.

In Figs. 19(d), 20(d) and 21(d) response time maintains almost stable through generations, because response time conflicts with all other objectives and Cielo trends to balance the trade-off among all the objectives. Cielo HV also reaches the best performance among all three Cielo variants with 26.11 ms of average in the last generation.

Table 5 compares Cielo three variants with two well-known heuristics algorithms, FFA (first-fit algorithm) and BFA (best-fit algorithm), which have been widely used for VM placement in clouds [LQL13, MLL12, GP13, CSV12]. The table shows the minimum, average and maximum objective values in the last generation. In all objectives, Cielo HV outperforms Cielo PD and Cielo HV-PD. The largest difference is in the minimum bandwidth allocation with DVFS disabled (40%), and the smallest difference is in the maximum response time with DVFS enabled (16.60%). FFA yields the lowest power consumption because it is designed to deploy VMs on the minimum number of hosts, however it sacrifices the other objectives. Theoretically BFA should performs the best in CPU allocation because it is designed to deploy VMs on the hosts that maintain higher resource availability. However Cielo HV is able to find the dominant strategy which distributes CPU allocation among hosts even better than BFA. Cielo maintains balanced objective values in between

FFA and BFA while Cielo yields the best performance in response time, CPU allocation and bandwidth allocation.

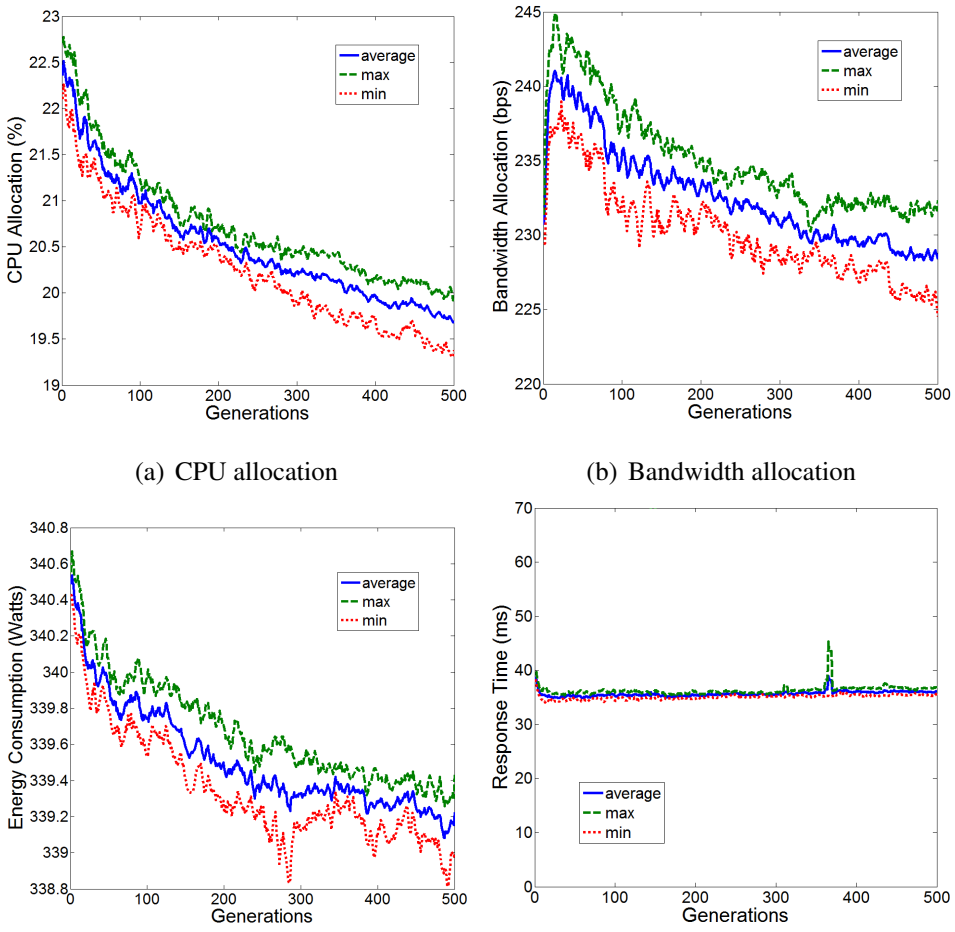
Table 4 shows the time required for Cielo three variants to execute given numbers of generations. Simulations were carried out with a Java VM 1.7 on a Windows 8.1 PC with a 3.6 GHz AMD A6-5400K APU and 6 GB memory space. For running a single simulation (i.e., 500 generations), Cielo HV runs 6 min 15 sec which is the fastest among all Cielo variant.

Fig. 22 illustrates how Cielo three variants evolve their hypervolume value through generations. Hypervolume value is the average computed using each application's dominant strategy in each generation. The results confirms again Cielo HV outperforms its hypervolume performance among other Cielo variants.

From simulation results, Cielo HV outperforms over other two Cielo variants in all objectives performance value and execution time. Cielo PD and Cielo HV-PD use the notion of pareto dominance, which requires to make multi comparison among all objectives. Cielo HV instead uses just one comparison to decide the winner. Pareto dominance asks for the strictly dominant strategy, one strategy should outperforms in all objectives and survives through generations in order to become the dominant strategy. However in most of the cases strategies are tie using pareto dominance because objectives are conflicting with each other.

4.4.4 Conclusion

This section describes and evaluates Cielo, a multiobjective evolutionary game theoretic framework for adaptive and stable application placement in clouds that support a power capping mechanism for CPUs. It aids cloud operators to schedule resources to applications and place applications onto particular CPU cores according to the operational conditions



(a) CPU allocation

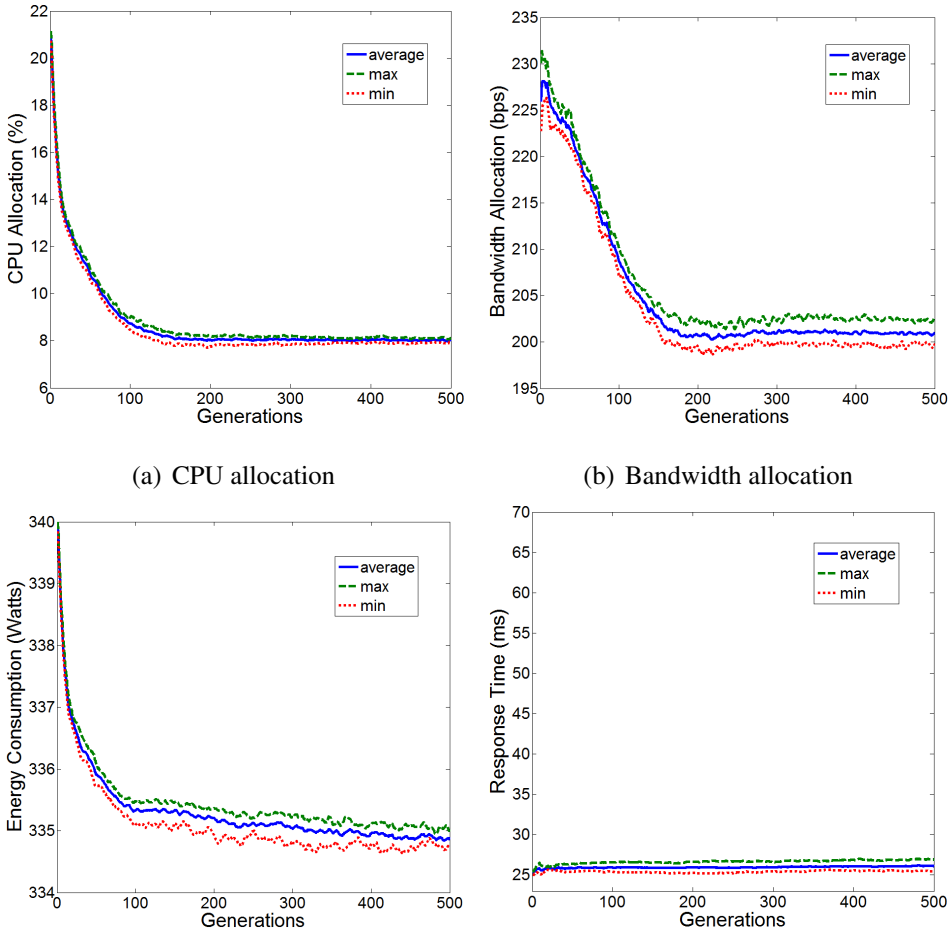
(b) Bandwidth allocation

(c) Energy consumption

(d) Response time

Figure 19: Cielo Pareto Dominance

in a cloud. It is theoretically guaranteed that Cielo allows each application to perform an evolutionarily stable deployment strategy, which is an equilibrium solution under given operational conditions. Simulation results verify that Cielo performs application deployment in an adaptive and stable manner. Cielo outperforms existing well-known heuristics: FFA an BFA.



(a) CPU allocation

(b) Bandwidth allocation

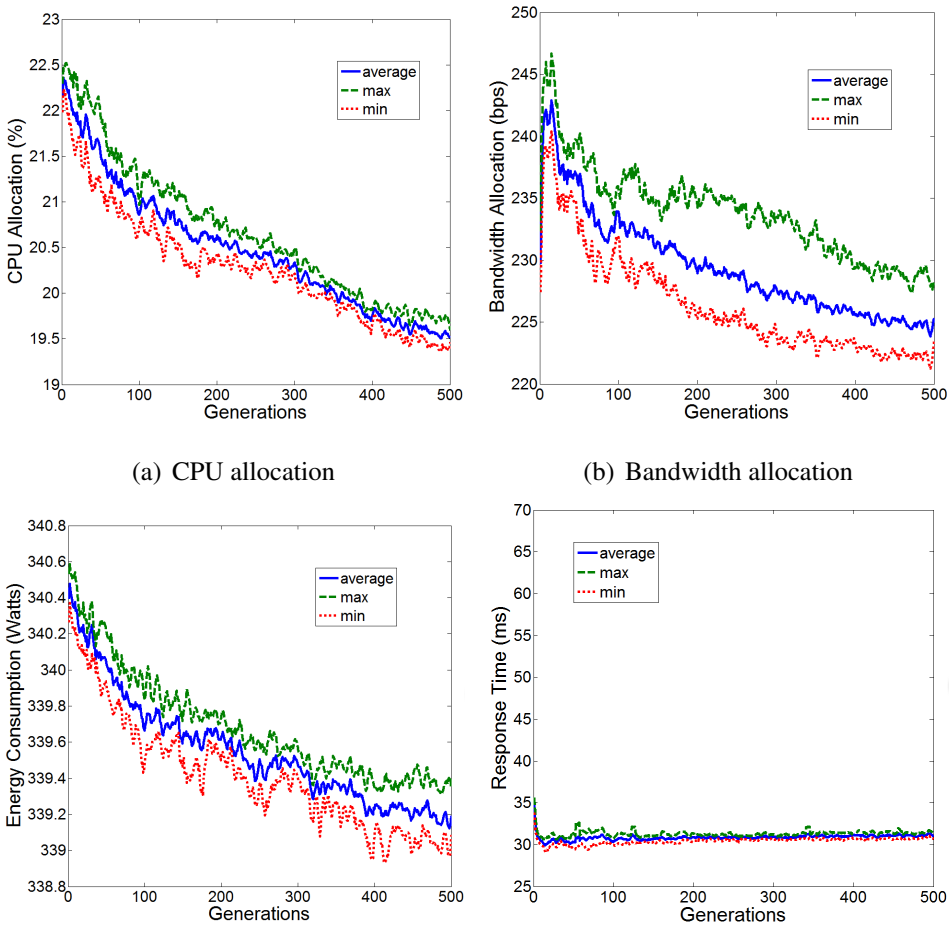
(c) Energy consumption

(d) Response time

Figure 20: Cielo Hypervolume

4.5 AGEGT

This section investigates a multiobjective evolutionary game theoretic framework for adaptive and stable application deployment in clouds that support dynamic voltage and frequency scaling (DVFS) for CPUs. The proposed algorithm, called AGEGT, aids cloud operators to adapt the resource allocation to applications and their locations according to the operational conditions in a cloud (e.g., workload and resource availability) with respect



(a) CPU allocation

(b) Bandwidth allocation

(c) Energy consumption

(d) Response time

Figure 21: Cielo Hypervolume & Pareto Dominance

to multiple conflicting objectives such as response time, resource utilization and power consumption. In AGEGT, evolutionary multiobjective games are performed on application deployment strategies (i.e., solution candidates) with an aid of guided local search. AGEGT theoretically guarantees that each application performs an evolutionarily stable deployment strategy, which is an equilibrium solution under given operational conditions. Simulation results verify this theoretical analysis; applications seek equilibria to perform adaptive and evolutionarily stable deployment strategies. AGEGT allows applications to

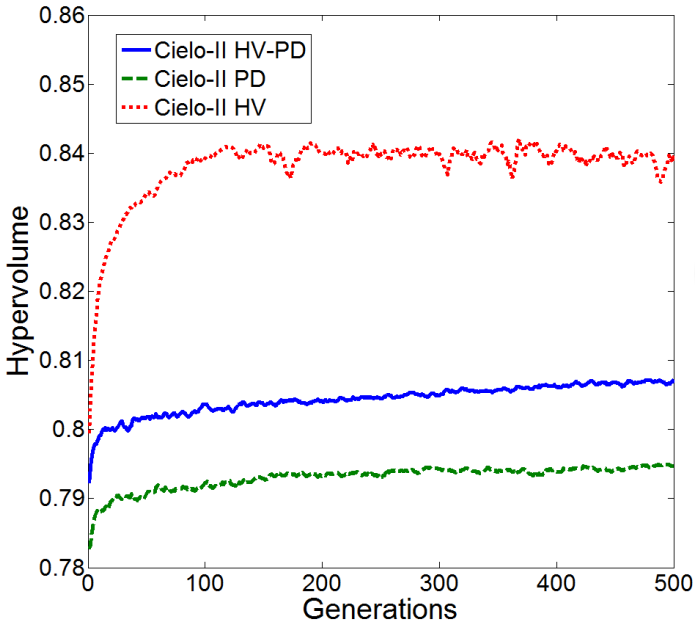


Figure 22: Cielo Hypervolume Comparison

successfully leverage DVFS to balance their response time, resource utilization and power consumption. AGEGT gains performance improvement via guided local search and outperforms existing heuristics such as first-fit and best-fit algorithms (FFA and BFA) as well as NSGA-II.

4.5.1 Introduction

AGEGT is an evolutionary game theoretic framework for adaptive and stable application deployment in clouds that support dynamic voltage and frequency scaling (DVFS) for CPUs. This section describes its design and evaluates its optimality and stability. In AGEGT, each application maintains a set (or a population) of deployment strategies, each of which indicates the location of and resource allocation for that application. AGEGT repeatedly performs evolutionary multiobjective games on deployment strategies and evolves them over generations with respect to conflicting objectives. In each generation, AGEGT

runs active-guided mutation, which alters deployment strategies based on the guided local search (GLS) algorithm [VT99]. It records inferior deployment strategies as penalties through generations and uses the penalties to help strategies escape from local optima and gain performance improvement.

AGEGT theoretically guarantees that, through a series of evolutionary games between deployment strategies, the population state (i.e., the distribution of strategies) converges to an evolutionarily stable equilibrium, which is always converged to regardless of the initial state. (A dominant strategy in the evolutionarily stable population state is called an evolutionarily stable strategy.) In this state, no other strategies except an evolutionarily stable strategy can dominate the population. Given this theoretical property, AGEGT aids each application to operate at equilibria by using an evolutionarily stable strategy for application deployment in a deterministic (i.e., stable) manner.

Simulation results verify this theoretical analysis; applications seek equilibria to perform evolutionarily stable deployment strategies and adapt their locations and resource allocations to given operational conditions. AGEGT allows applications to successfully leverage DVFS to balance their response time performance, resource utilization and power consumption. AGEGT's performance is evaluated in comparison to existing heuristic algorithms. Simulation results demonstrate that AGEGT yields a 30.3x speedup against a well-known multiobjective evolutionary optimization algorithm, NSGA-II [DAP00], in convergence speed and maintains 98% higher stability (lower oscillations) in performance across different simulation runs. Moreover, AGEGT outperforms two well-known heuristics, first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [LQL13, MLL12, GP13, CSV12].

4.5.2 Algorithm

AGEGT maintains N populations, $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$, for N applications and performs games among strategies in each population. A strategy s consists of five parameters to indicate the locations of and the resource allocation for three VMs in a particular application:

$$s(a_i) = \bigcup_{t \in 1,2,3} \left(h_{i,t},\ u_{i,t},\ c_{i,t},\ b_{i,t},\ q_{i,t} \right),\ \ 1 < i < N \tag{4.14}$$

a_i denotes the i-th application. $h_{i,t}$ is the ID of a host that a_i's t-th tier VM is placed to. $u_{i,t}$ is the ID of a CPU core that a_i's t-th tier VM resides on in the host $h_{i,t}$. $c_{i,t}$ and $b_{i,t}$ are the CPU and bandwidth allocation for a_i's t-th tier VM. $q_{i,t}$ denotes the frequency of a CPU core that a_i's t-th tier VM resides on.

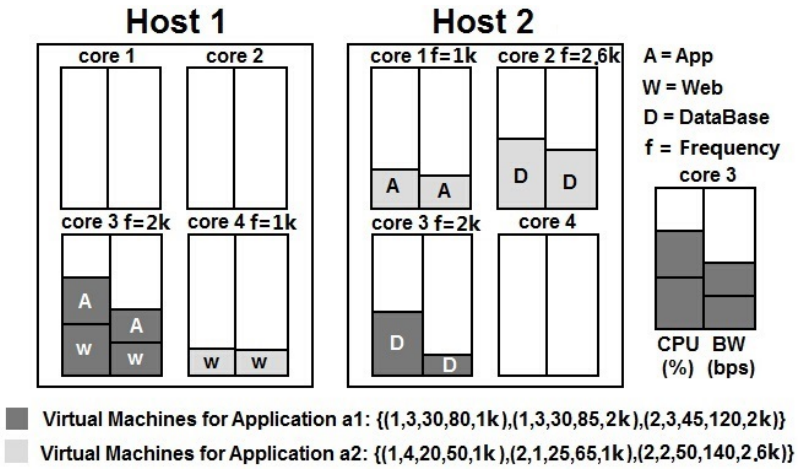


Figure 23: AGEGT Example Deployment Strategies

Fig. 23 shows two example strategies for two applications (a_1 and a_2) ($N = 2$). Four cores are available in each of two hosts ($M = 2$ and $O = 4$). a_1's strategy, $s(a_1)$, places the first-tier VM on the third core in the first host ($h_{1,1} = 1$ and $u_{1,1} = 3$). The 30% time share

of the CPU core and 80 Kbps bandwidth are allocated to the VM ($c_{1,1} = 30$ and $b_{1,1} = 80$). The VM requires the frequency of 1 GHz for the CPU core ($q_{1,1} = 1k$). The second-tier VM of a_1 is placed on the third core in the first host ($h_{1,2} = 1$, $u_{1,2} = 3$). 30% of the CPU core time and 85 Kbps bandwidth are allocated to the VM ($c_{1,2} = 30$ and $b_{1,2} = 85$). The VM requires the frequency of 2 GHz for the CPU core ($q_{1,2} = 2k$). The third-tier VM of a_1 requires the frequency of 2 GHz ($q_{1,3} = 2k$) on the third core of the second host ($h_{1,3} = 2$, $u_{1,3} = 3$). 45% of the CPU core time and 120 Kbps bandwidth are allocated to the VM ($c_{1,3} = 45$ and $b_{1,3} = 120$). If multiple VMs are placed on a CPU core, the core operates at the highest required frequency. For example, on the third core of the first host, two VMs requires 1 GHz and 2 GHz. Thus, the core operates at 2 GHz. Note that the left and right columns of each CPU core denote the CPU share and bandwidth allocation for the VMs placed on that CPU core.

Given $s(a_1)$, a_1's objective values for CPU and bandwidth allocation are 105% (30 + 30 + 45) and 285 kbps (80 + 85 + 120). Assuming the CPU core capacity constraint $C_C = 100\%$ (Equation 4.8), it is satisfied on every core ($g_C = 0$). For example, on the third core of the first host, the total share allocation $c_{1,3}$ is 60% (30% + 30%).

Algorithm 4.5.1 shows how AGEGT seeks an evolutionarily stable strategy for each application through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$ (Line 2). Those strategies may or may not be feasible. Note that a strategy is said to be feasible if it violates none of four constraints described in Section 4.3. A strategy is said to be infeasible if it violates at least one constraint.

In each generation (g), a series of games are carried out on every population (Lines 4 to 28). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to the objectives described in Section 4.3

(Lines 7 to 9). A game is carried out based on the superior-inferior relationship between the two strategies and their feasibility (c.f. performGame() in Algorithm 4.5.1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins over its opponent. If both strategies are feasible, they are compared with the Hypervolume (HV) metric [ZT98]. It measures the volume that a given strategy s dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{4.15}$$

Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. The notion of Pareto dominance ($\succ$) is defined as follows. A strategy s_1 is said to dominate another strategy s_2 ($s_1 \succ s_2$) if both of the following conditions hold:

- s_1's objective values are superior than, or equal to, s_2's in all objectives.
- s_1's objective values are superior than s_2's in at least one objectives.

A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. An infeasible strategy s_1 wins a game over another infeasible strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

Once a game determines the winner and the loser, the winner replicates itself (Line 10). The replica is altered through active-guided mutation, which mutates a strategy with guided local search (GLS) [VT99] (Lines 11 to 16).

4.5.2.1 Active Guided Approach

From three VMs in an application that a given strategy represents ($a_{i,t} \in a_i$), Active Guided scheme first identifies the VM that yields the worst performance with modified objective functions (Line 11) and then mutates a parameter(s) for the VM (Lines 12 to 16). Eq. 4.16 is used to evaluate the t-th VM of an application that a strategy s is represents.

$$\vec{u_t}(s) = \frac{\vec{f_t'}(s)}{1+\varphi_k} \tag{4.16}$$

$\vec{f_t'}(s) = \{f'_{t,C}(s), f'_{t,B}(s), f'_{t,RT}(s), f'_{t,PC}(s)\}$ is a vector of modified objective functions. For example, $f'_{t,C}(s)$ is computed based on the CPU allocation for the t-th VM of an application that a strategy s represents. Modified objective functions are defined as follows.

$$\vec{f_t'}(s) = \vec{f_t}(s) + \eta \sum_{k=1}^{U} \varphi_k I_{k,t}(s) \tag{4.17}$$

Each function $f_t(s)$ is computed on a VM by VM basis by customizing the original objective function (Eq. 4.1, 4.2, 4.3 or 4.7). For example, $f_{t,C}(s)$ indicates the CPU allocation for the t-th VM of an application that s represents. U denotes the total number of CPU cores in a cloud: $U = M * O$. φ_k is the penalty for the k-th CPU core. It is initialized as 0 in the first generation and incremented when the worst VM resides on the k-th CPU core. $I_k(s)$ is a boolean variable that contains 1 if the k-th CPU core is assigned to the t-th VM of an application that s represents and otherwise 0. η is a constant, which is called penalty rate.

Active-guided mutation evaluates the performance of each VM of an application that a strategy s represents, as $\vec{u}_t(s)$ in Eq. 4.16, and determines the worst VM by comparing $\vec{u}_t(s)$, $1 \leq t \leq 3$, with the hypervolume metric (Eq. 4.15).

The worst VM is chosen in Line 11. In Eq. 4.16, k in P_k indicates the CPU core that the t-th VM resides on, and φ_k denotes the total amount of penalty that the CPU core has accumulated in the past generations.

Active-guided mutation also uses $\vec{u}_t(s)$ in Eq. 4.16 as the utility of penalizing each VM. Once it determines the worst VM in Line 11, it increments the penalty for the CPU core that the VM resides on (φ_k in Eq. 4.16). In Lines 12 to 16, it randomly chooses a parameter (or parameters) of the worst VM identified in Line 11 with a certain mutation rate P_m and alters its/their value(s) at random based on polynomial mutation [DPA02]. (mutate() in Line 14 implements polynomial mutation.) Key ideas behind active-guided mutation are to (1) identify the worst-performing VM in each application and alter its deployment strategy to potentially improve its performance and (2) record inferior VM deployment strategies as penalties through generations and use the penalties to help strategies escape from local optima and improve their performance.

Mutation is followed by a game performed between the loser and the mutated winner (Line 17). This is intended to select the top two of three strategies (the winner, loser and mutated winner). The worst of the three strategies disappears in the population.

Once all strategies play games in the population, AGEGT identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 22 to 26). AGEGT uses the dominant strategy to adjust the parameters for three VMs of an application in question (Line 27).

4.5.3 Experiment

This section evaluates AGEGT's performance, particularly in its optimality and stability, through simulations.

4.5.3.1 Experiment Setting

Experiment simulates a cloud data center that consists of 100 hosts in a 10×10 grid topology. The grid topology is chosen based on recent findings on efficient topology configurations in clouds [GWT08, GLL09]. It also assumes five different types of applications. Table 6 shows the message arrival rate (i.e., the number of incoming messages per second) and message processing time (in second) for each of the five application types. This configuration follows Zipf's law [Per96, THL03]. This experiment simulates 40 application instances for each type (200 application instances in total).

Each host is simulated to operate an Intel Core2 Quad Q6700 CPU, which is a quad-core CPU that has five frequency and voltage operating points (P-states). Table 7 shows the power consumption at each P-state under the 0% and 100% CPU utilization [GMC13]. This setting is used in Equation 4.7 to compute power consumption objective values.

Table 8 shows the parameter settings for AGEGT. Mutation rate is set to $1/v$ where v is the number of parameters in a strategy. ($v = 5$ as shown in Eq. 4.14). Every simulation result is the average with 20 independent simulation runs.

Comparative performance study is carried out for AGEGT and its two variants: EGT-GLS and EGT. Algorithm 4.5.2 shows the procedure of EGT-GLS, which is similar to AGEGT. EGT-GLS performs polynomial mutation, instead of active-guided mutation, and guided local search (GLS) in each generation (i.e., localSearch() in Algorithm 4.5.2). The local search operator is designed to improve the performance of a dominance strategy d_i (Algorithm 4.5.3). It creates Q mutants of d_i iteratively using GLS and replaces d_i with a mutant if the mutant wins over d_i in a game. Through Q iterations, the local search operator keeps the best mutant discovered so far and mutates it when mutation occurs. Another variant, EGT, performs Algorithm 4.5.2 with local search disabled.

AGEGT is also compared with NSGA-II, which is a well-known multiobjective evolutionary algorithm [DPA02]. AGEGT (and its variants) and NSGA-II use the same parameter settings shown in Table 8. All other NSGA-II settings are borrowed from [DPA02]. AGEGT, EGT-GLS, EGT and NSGA-II are implemented with jMetal [DNA10]. Moreover, AGEGT is evaluated in comparison

to well-known heuristics, first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [LQL13, MLL12, GP13, CSV12].

Table 9 shows two different combinations of constraints: no constraints (C_∞) and moderate (C_M). C_M is used unless otherwise noted.

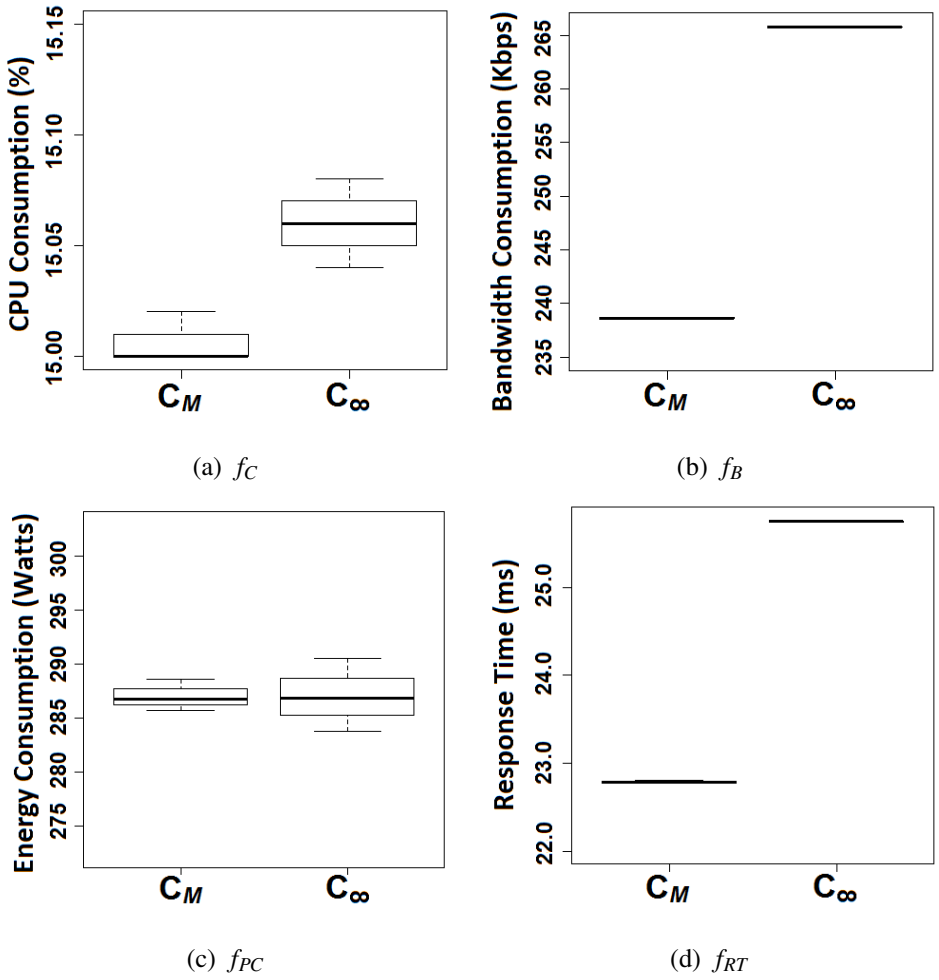
4.5.3.2 Experiment Results

Table 10 examines how a mutation-related parameter, called distribution index (η_m in [DPA02]), impacts the performance of EGT. This parameter controls how likely a mutated strategy is similar to its original. (A higher distribution index makes a mutant more similar to its original.) In Table 10, the performance of EGT is evaluated with the hypervolume (HV) measure that a set of dominant strategies yield in the 500th generation. A higher HV means that a set of solutions is more optimal (c.f. Eq. 4.15). As shown in Table 10, EGT yields the best performance with the distribution index value of 40.

Thus, this parameter setting is used for EGT, EGT-GLS and AGEGT in all successive simulations.

Figs. 24 to 26 show a series of boxplots for the objective values that AGEGT, EGT-GLS and EGT yield in the last generation under two different constraints combination (C_∞ and C_M). The ends of whiskers indicates the maximum and minimum objective values. All three algorithms perform better under constraints. This means that constraint handling works properly in games. All successive simulations use the moderate constraint combination (C_M). AGEGT often yields lower variance in objective values than EGT-GLS and EGT do. This means AGEGT performs more consistently among different simulation runs. In comparison to EGT-GLS and EGT, AGEGT outperforms them in two and three objectives, respectively.

Fig. 27 shows how AGEGT, EGT-GLS and EGT evolve their HV through generations. The highest HV that the three algorithms yield are 0.835, 0.832 and 0.830, respectively. In HV, AGEGT gains 0.3% and 0.6% improvements over EGT-GLS and EGT. This result is consistent with the ones



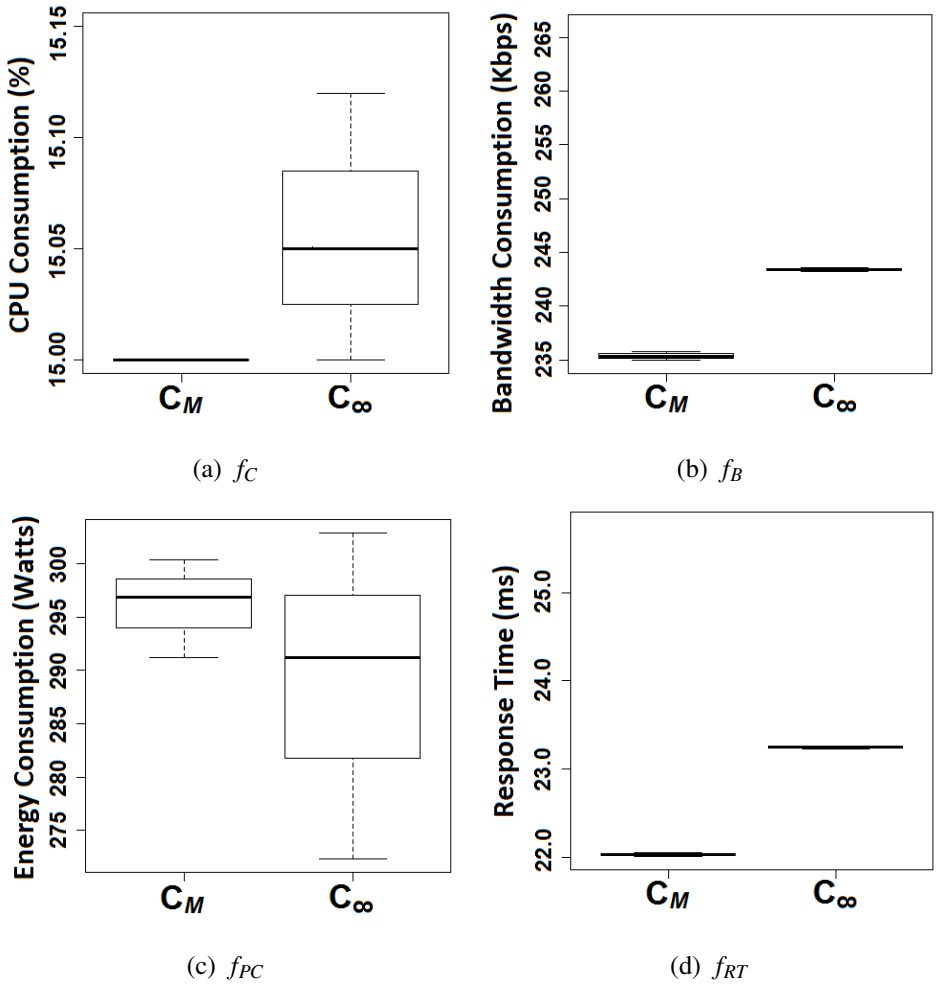
(a) f_C (b) f_B

(c) f_{PC} (d) f_{RT}

Figure 24: Objective Values of AGEGT under Two Constraint Combinations

in Figs. 24 to 26. Table 11 depicts how many generations AGEGT, EGT-GLS and EGT require to reach the HV values of 0.835, 0.832 and 0.830. In order to reach the HV value of 0.832, AGEGT gains a 5.6x speedup over EGT-GLS. It gains a 4.7x speed up over EGT to reach the HV value of 0.830. Fig. 27 and Table 11 demonstrate that AGEGT significantly outperforms EGT-GLS and EGT in convergence speed while its superiority is not significant in optimality. Active-guided mutation aids AGEGT to gain performance improvement effectively.

In comparison of AGEGT with NSGA-II, the set of AGEGT dominant strategies at the last generation is non-dominated (i.e., tie) with the NSGA-II individuals at the last generation based



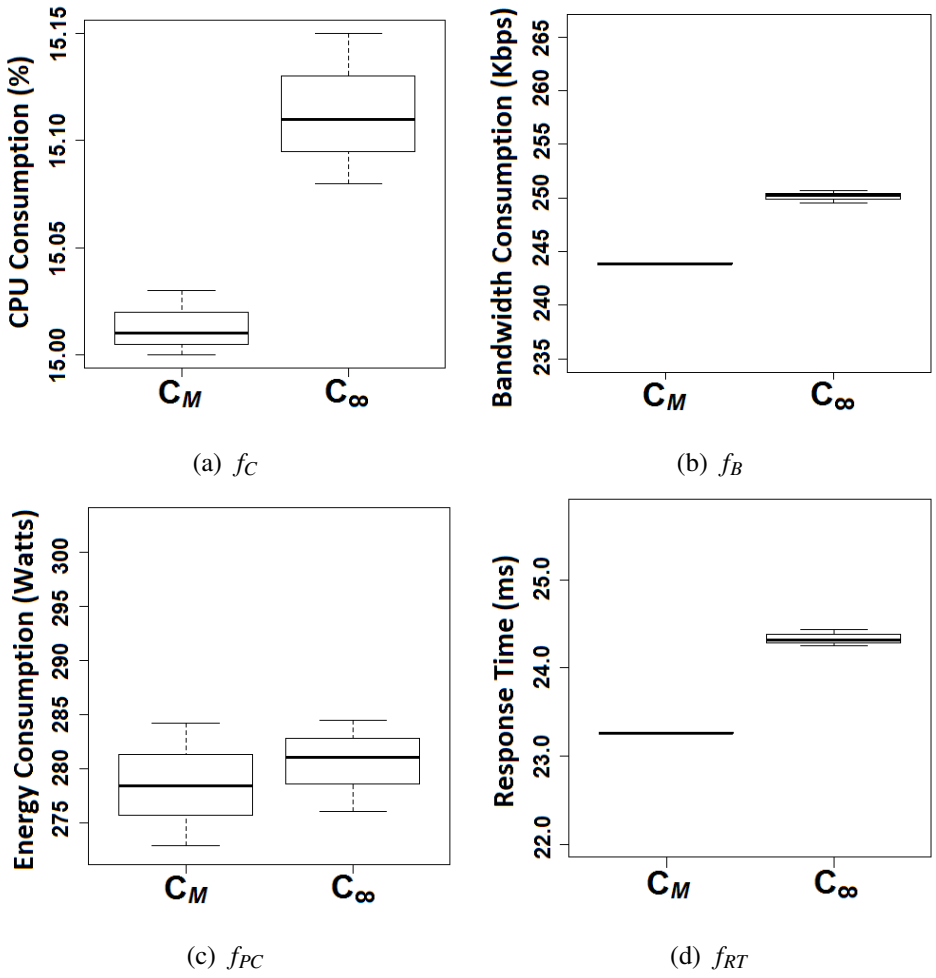
(a) f_C (b) f_B

(c) f_{PC} (d) f_{RT}

Figure 25: Objective Values of EGT-GLS under Two Constraint Combinations

on the notion of Pareto dominance. While AGEGT yields the HV value of 0.835, the highest HV value that NSGA-II yields is 0.813. (The difference is 2.70%.) NSGA-II requires 182 generations to reach the HV value of 0.813 (Table 11). In contrast, AGEGT spends only 6 generations to reach the HV value. It maintains a 30.3x speedup against NSGA-II in convergence speed.

Table 12 examines the optimality of AGEGT and NSGA-II in distance metrics. It shows the distance in the normalized objective space from the Utopian point, (0, 0, 0, 0), to the set of dominant strategies that AGEGT produces at the last generation. In its middle column, it shows the average distance from the Utopian point to the individuals that NSGA-II produces at the last generation.



(a) f_C (b) f_B

(c) f_{PC} (d) f_{RT}

Figure 26: Objective Values of EGT under Two Constraint Combinations

Euclidean and Manhattan distances are used as metrics. In both metrics, a shorter distance means a set of solutions are closer to the Utopian point (i.e., more optimal). The right column shows the distance from the Utopian point to one of NSGA-II individuals that is closest to the set of AGEGT strategies at the last generation. Table 12 shows that AGEGT is closer to the Utopian point than NSGA-II by 32% to 47%.

Table 13 compares AGEGT with NSGA-II, FFA and BFA. It shows the minimum, average and maximum objective values in the last generation. AGEGT outperforms NSGA-II in the average CPU allocation, bandwidth consumption and power consumption by 50.37%, 19.28% and 77.63%,

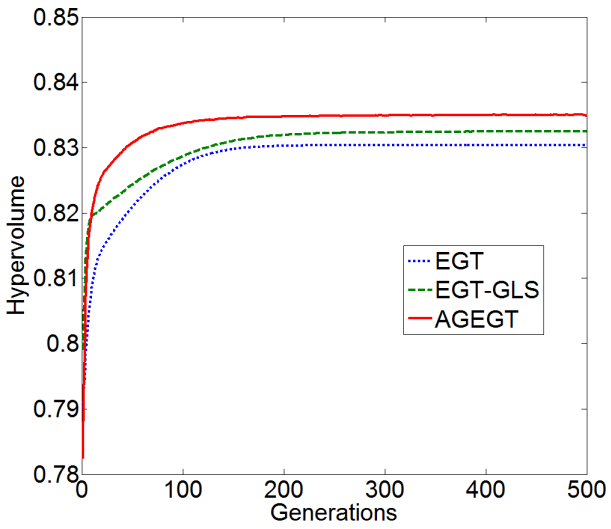
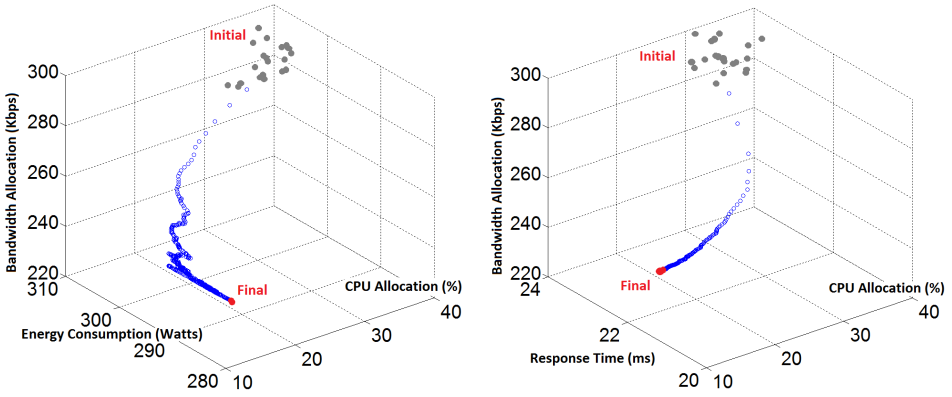


Figure 27: Comparison of AGEGT, EGT-GLS and EGT in Hypervolume (HV)

respectively. In response time, NSGA-II outperforms AGEGT by 47.17%. On average, AGEGT outperforms NSGA-II by 25.02%. FFA and BFA produce two extreme results. FFA yields the lowest power consumption (59.61 Watts) because it is designed to deploy VMs on the minimum number of hosts; however, it sacrifices the other objectives. BFA performs the best in CPU allocation (28.94 %) because it is designed to deploy VMs on the hosts that maintain higher resource availability. AGEGT maintains balanced objective values in between FFA and BFA while performing better in response time, CPU allocation and bandwidth allocation.

Table 14 shows the variance of objective values that AGEGT and NSGA-II yield at the last generation in 20 different simulation runs. A lower variance means higher stability (or higher similarity) in objective value results (i.e., lower oscillations in objective value results) among different simulation runs. AGEGT maintains significantly higher stability than NSGA-II in all objectives except response time. AGEGT's average stability is 98.86% higher than NSGA-II's. This result exhibits AGEGT's stability property (i.e. ability to seek evolutionarily stable strategies), which NSGA-II does not have.

Fig. 28 shows two three-dimensional objective spaces that plot a set of dominant strategies obtained from individual populations at each generation. Each blue dot indicates the average objective values that dominant strategies yield at a particular generation in 20 simulation runs. The trajectory of blue dots illustrates a path through which AGEGT's strategies evolve and improve objective values. Gray and red dots represent 20 different sets of objective values at the first and last generation in 20 simulation runs, respectively. While initial (gray) dots disperse (because strategies are generated at random initially), final (red) dots are overlapped in a small region. Consistent with Table 14, Fig. 28 verifies AGEGT's stability: reachability to at least one Nash equilibria regardless of the initial conditions.



(a) CPU allocation, Bandwidth allocation and Energy consumption

(b) CPU allocation, Bandwidth allocation and Response time

Figure 28: Trajectory of AGEGT's Solution through Generations

4.5.4 Conclusion

This section proposes and evaluates AGEGT, an evolutionary game theoretic framework for adaptive and stable VM deployment in DVFS-enabled clouds. It theoretically guarantees that every application (i.e., a set of VMs) seeks an evolutionarily stable deployment strategy, which is an equilibrium solution under given workload and resource availability. Simulation results verify that AGEGT performs VM deployment in an adaptive and stable manner. AGEGT outperforms existing well-known

heuristics in the quality, efficiency and stability of VM deployment. For example, AGEGT yields a 30.3x speedup against NSGA-II in convergence speed and maintains 98% higher stability.

4.6 Cielo-LP

In this section, evolutionary multi-objective games are performed on VM configuration strategies (i.e., solution candidates) with an aid of linear programming. Cielo-LP theoretically guarantees that each application (i.e., a set of VMs) performs an evolutionarily stable deployment strategy, which is an equilibrium solution under given operational conditions. Simulation results verify this theoretical stability analysis, and applications seek equilibria to perform adaptive and evolutionarily stable deployment strategies. Linear programming allows Cielo-LP to gain up to 38% improvement in optimality and up to 5.5x speedup in convergence speed with reasonably acceptable computational costs.

4.6.1 Introduction

In Cielo-LP, each application maintains a set (or a population) of placement strategies, each of which indicates the location of and resource allocation for that application. Cielo-LP repeatedly performs evolutionary multiobjective games on placement strategies and evolves them over generations with respect to conflicting optimization objectives including response time, resource utilization and power consumption. In each generation, Cielo-LP runs the simplex linear programming (LP) algorithm for a small portion of the entire search space and leverages the LP-optimal local solution(s) to efficiently search a globally optimal solution.

This section describes Cielo-LP's algorithmic design and evaluates its optimality and stability with a cloud data center that supports dynamic voltage and frequency scaling (DVFS) for CPUs. Simulation results demonstrate that Cielo-LP allows applications to seek equilibria to perform evolutionarily stable placement strategies and adapt their locations and resource allocations to given operational conditions.

To the best of my knowledge, this study is the first attempt to integrate linear programming (LP) with an EGT-backed evolutionary algorithm. There exist a few research efforts to integrate LP with traditional evolutionary algorithms (e.g., [GN07, GKB11]). Cielo-LP is similar to [GN07] in that both work use LP to solve a part of the entire problem and approach the rest of the problem with an evolutionary algorithm based on LP-optimal solutions. In [GKB11], Kumar et al. solve a relaxed version of the problem with LP and use an evolutionary algorithm for the complete problem based on the LP-optimal solutions. Both of the two relevant work focus on optimization only, not stability, and consider a single objective, while Cielo-LP considers both optimization and stability with respect to multiple objectives.

4.6.2 Algorithm

Cielo-LP maintains N populations, $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$, for N applications and performs games among strategies in each population. A strategy s consists of five parameters to indicate the locations of and the resource allocation for three VMs in a particular application:

$$s(a_i) = \bigcup_{t \in 1,2,3} \left(h_{i,t},\ u_{i,t},\ c_{i,t},\ b_{i,t},\ q_{i,t} \right),\ \ 1 < i < N \tag{4.18}$$

a_i denotes the i-th application. $h_{i,t}$ is the ID of a host that a_i's t-th tier VM is placed to. $u_{i,t}$ is the ID of a CPU core that a_i's t-th tier VM resides on in the host $h_{i,t}$. $c_{i,t}$ and $b_{i,t}$ are the CPU and bandwidth allocation for a_i's t-th tier VM. $q_{i,t}$ denotes the frequency of a CPU core that a_i's t-th tier VM resides on.

Fig. 29 shows two example strategies for two applications (a_1 and a_2) ($N = 2$). Four cores are available in each of two hosts ($M = 3$ and $O = 2$). a_1's strategy, $s(a_1)$, places the first-tier VM on the third core in the first host ($h_{1,1} = 1$ and $u_{1,1} = 3$). The 30% time share of the CPU core and 80 Kbps bandwidth are allocated to the VM ($c_{1,1} = 30$ and $b_{1,1} = 80$). The VM requires the frequency of 1 GHz for the CPU core ($q_{1,1} = 1k$). The second-tier VM of a_1 is placed on the third core in the first host ($h_{1,2} = 1$, $u_{1,2} = 3$). 30% of the CPU core time and 85 Kbps bandwidth are allocated

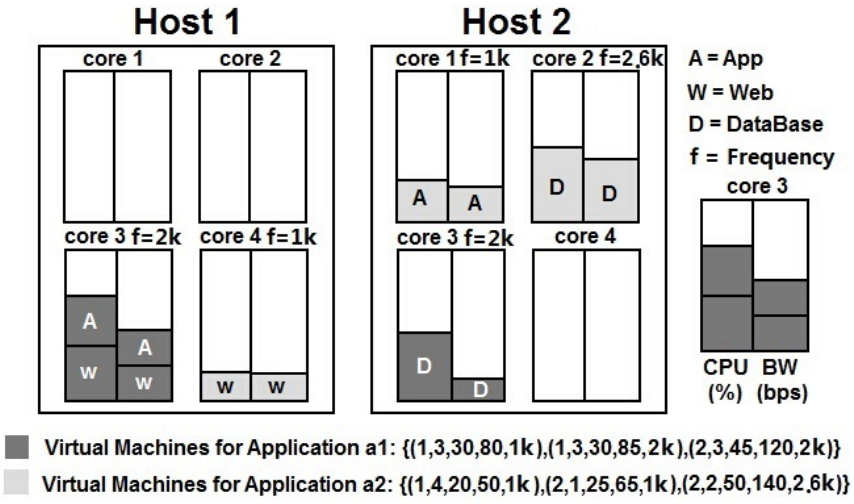


Figure 29: Cielo-LP Example Deployment Strategies

to the VM ($c_{1,2} = 30$ and $b_{1,2} = 85$). The VM requires the frequency of 2 GHz for the CPU core ($q_{1,2} = 2k$). The third-tier VM of a_1 requires the frequency of 2 GHz ($q_{1,3} = 2k$) on the third core of the second host ($h_{1,3} = 2$, $u_{1,3} = 3$). 45% of the CPU core time and 120 Kbps bandwidth are allocated to the VM ($c_{1,3} = 45$ and $b_{1,3} = 120$). If multiple VMs are placed on a CPU core, the core operates at the highest required frequency. For example, on the third core of the first host, two VMs requires 1 GHz and 2 GHz. Thus, the core operates at 2 GHz.

Given $s(a_1)$, a_1's objective values for CPU and bandwidth allocation are 105% (30 + 30 + 45) and 285 kbps (80 + 85 + 120). Assuming the CPU core capacity constraint $C_C = 100\%$ (Eq. 4.8), it is satisfied on every core ($g_C = 0$). For example, on the third core of the first host, the total share allocation $c_{1,3}$ is 60% (30% + 30%).

Algorithm 4.6.1 shows how Cielo-LP seeks an evolutionarily stable strategy for each application through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$ (Line 1). Those strategies may or may not be feasible. Note that a strategy is said to be feasible if it violates none of four constraints described in Section 4.3. A strategy is said to be infeasible if it violates at least one constraint.

In each generation (g), under the probability of $1 - P_l$, a series of games are carried out on every population (Lines 3 to 30). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to performance objectives described in Section 4.3 (Lines 11 to 12). The loser disappears in the population. The winner is replicated to increase its population share and mutated with polynomial mutation [DPA02] (Lines 13 to 18).

Mutation randomly chooses a parameter (or parameters) in a given strategy with a certain mutation rate P_m and alters its/their value(s) at random (Lines 14 to 18). Then, another game is performed between the loser and the mutated winner (Line 19). This is intended to select the top two of three strategies (winner, loser and mutated winner).

Once all strategies play games in the population, Cielo-LP identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 24 to 27). Cielo-LP deploys three VMs for an application in question based on the dominant strategy (Line 28).

A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (performGame() in Algorithm 4.6.1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins the game. If both strategies are feasible, they are compared based on the notion of Pareto dominance [SD95], in which a strategy s_1 is said to dominate another strategy s_2 if both of the following conditions hold:

- s_1's objective values are superior than, or equal to, s_2's in all objectives.
- s_1's objective values are superior than s_2's in at least one objectives.

The dominating strategy wins a game over the dominated one. If two strategies are non-dominated with each other, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. An infeasible strategy s_1 wins a game over another infeasible strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.

- s_1's constraint violation is lower than s_2's in at least one constraints.

4.6.2.1 Linear Programming Integration

In each generation (g), Cielo-LP performs the simplex linear programming (LP) algorithm on each population ($\mathscr{P}_i$) with the probability of P_l (Lines 6 to 7). LP guarantees to find the optimal solution for a single objective function as far as it exists; however, it cannot consider multiple objectives separately. Therefore, Cielo-LP executes LP in one of the following two methods subject to the four constraints described in Section 4.3 (C_C,C_B,C_{RT} and C_{PC}).

- Single objective method: Cielo executes LP with one of four objectives described in Section 4.3 (f_C, f_B, f_{RT} or f_{PC}). Another limitation of LP is that objective and constraint functions must be all linear. Since the objective function for response time is non-linear (Eq. 4.3), Cielo replaces it with a linearly approximated function:

$$f_{RT}^{LP} = \sum_{t=1}^{3} \left\{ T_t^p + \frac{1}{O}(T_t^p \lambda_t - c_t \frac{q_t}{q_{max}}) + (B\lambda_t - b_t) \right\} \tag{4.19}$$

- Weighted sum method: Cielo-LP executes LP with the following objective function, which aggregates four objective functions (f_C, f_B, f_{RT}^{LP} or f_{PC}) as a weighted sum. w denotes a weight value for a particular objective function. Each objective value (f_i) is normalized.

$$f_{WS} = \sum_{i=1}^{4} w_i f_i \tag{4.20}$$

Once LP finds the optimal solution for a population, Cielo-LP treats it as the dominant strategy for that population (Line 7). Note that the linear programming rate (P_l) is intended to be low. Cielo-LP executes LP for a small portion of the entire problem (i.e., for a small number of populations: $P_l \times N$ populations) and leverages the LP-optimal solution(s) to boost convergence speed as well as the quality of the set of dominant strategies d_1, d_2, ... , d_N).

4.6.3 Experiment

This section evaluates Cielo-LP, particularly in its optimality and stability, through simulations.

4.6.3.1 Experiment Setting

Experiment simulates a cloud data center that consists of 100 hosts in a 10×10 grid topology ($M = 100$). The grid topology is chosen based on recent findings on efficient topology configurations in clouds [GWT08, GLL09]. It assumes five different types of applications. Table 4.6.3.1 shows the message arrival rate (i.e., the number of incoming messages per second) and message processing time (in second) for each type of applications. This configuration follows Zipf's law [Per96, THL03]. This experiment simulates 40 application instances for each application type (200 application instances in total; $N = 200$).

Each host is simulated to operate an Intel Core2 Quad Q6700 CPU, which is a quad-core CPU that has five frequency and voltage operating points (P-states). Table 16 shows the power consumption at each P-state under the 0% and 100% CPU utilization [GMC13]. This setting is used in Eq. 4.7 to compute power consumption objective values.

Table 17 shows the parameter settings for Cielo. Mutation rate is set to $1/v$ where v is the number of parameters in a strategy. ($v = 15$ as shown in Eq. 4.18). Every simulation result is the average with 20 independent simulation runs.

Comparative study is carried out for the following variants of Cielo-LP.

- Cielo-BASE: Cielo with linear programming (LP) disabled
- $Cielo - LP_C$: Cielo with LP that uses the CPU consumption objective
- $Cielo - LP_B$: Cielo with LP that uses the bandwidth consumption objective
- $Cielo - LP_{PC}$: Cielo with LP that uses the power consumption objective
- $Cielo - LP_{RT}$: Cielo with LP that uses the response time objective

- $Cielo - LP_{WS}$: Cielo with LP that uses a weighted sum of objective values as an objective (Eq. 4.20).

CieloLP's variants are compared with and without constraints (C_M and C_∞ in Table 18). Constraints are enabled unless otherwise noted.

Cielo-LP is evaluated in comparison with the simplex LP algorithm as well as NSGA-II, which is a well-known multiobjective evolutionary algorithm [DAP00]. Simplex is implemented with GNU Linear Programming Kit (GLPK)[1] and Xypron[2]. Cielo-LP and NSGA-II use the same parameter settings shown in Table 17. All other NSGA-II settings are borrowed from [DAP00]. Both Cielo-LP and NSGA-II are implemented with jMetal [DNA10]. Moreover, Cielo-LP is compared to well-known heuristics, first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [LQL13, MLL12, GP13, CSV12]. All simulations were carried out with a Java VM 1.7 on a Windows 8.1 PC with a 3.6 GHz AMD A6-5400K APU and 6 GB memory space.

4.6.3.2 Experiment Results

Table 19 examines how a mutation-related parameter, called distribution index (η_m in [DPA02]), impacts the performance of Cielo-LP. Cielo-BASE is used in this evaluation. This parameter controls how likely a mutated strategy is similar to its original. (A higher distribution index makes a mutant more similar to its original.) In Table 19, the performance of Cielo-LP is evaluated with the hypervolume measure that a set of dominant strategies yield in the last (500th) generation. The hypervolume metric indicates the union of the volumes that a given set of solutions dominates in the objective space [ZT98]. A higher hypervolume means that a set of solutions is more optimal. As shown in Table 19, Cielo-LP yields the best performance with the distribution index value of 40. Thus, this parameter setting is used in all successive simulations.

[1] http://www.gnu.org/software/glpk/
[2] http://glpk-java.sourceforge.net/

Table 20 illustrates how the computational costs of Cielo-LP changes according to the linear programming (LP) rate (P_l in Algorithm 4.6.1 and Table 17). When the LP rate is 0 %, Cielo-LP works as Cielo-BASE. Its execution time is approximately 6.5 minutes to run 500 generations. This is an acceptable cost for configuring 3,000 parameters for 200 applications (15 parameters/application $\times$ 200 applications).

Table 20 also shows the computational costs of $Cielo - LP_{WS}$ under different LP rates from 0.5% to 50%. Probabilistically, LP is applied on one of 200 applications in each generation under the LP rate of 0.5%. If $Cielo - LP_{WS}$ considers all four objectives in its weighted-sum function (Eq. 4.20), its execution time exceeds one hour even if the LP rate is 0.5%. The computation of response time is a major contribution to this cost. If $Cielo - LP_{WS}$ considers three objectives besides the response time objective, its computational costs are reasonably acceptable under the LP rates of 10% or lower. As LP rate increases from 0% to 10%, execution time approximately doubles; however, it is still less than 14 minutes. In all successive simulations, $Cielo - LP_{WS}$ does not consider the response time objective when it runs LP.

Figs. 30 to 33 show a series of boxplots for the objective values that Cielo-BASE and $Cielo - LP_{WS}$ yield in 20 simulation runs. Each boxplot illustrates the maximum and minimum objective values as well as the first, second and third quartiles of objective values. $Cielo - LP_{WS}$ outperforms Cielo-BASE in all objectives except the response time objective. Note that $Cielo - LP_{WS}$ does not consider the response time objective when it runs LP. $Cielo - LP_{WS}$ yields better/lower objective values as its LP rate increases from 0.5% to 10%. Since the computational cost is reasonably acceptable under the LP rate of 10% (Table 20), LP rate is set to 10% in all successive simulations.

Table 21 shows the average objective value that each of Cielo-LP's variants yields in the last generation. A number in parentheses indicates a performance gain against Cielo-BASE. Cielo always gains performance improvement on an objective(s) that LP is applied to. $Cielo - LP_C$ gains the highest performance improvement (38.6%). The performance improvement of $Cielo - LP_{WS}$ is 13.3% on average. Table 21 also depicts the distance from Cielo-LP's solution to the utopian point, which is (0, 0, 0, 0), in the objective space. Manhattan distance is used as a distance metric here.

$Cielo - LP_{WS}$'s solution is 13.2% closer to the utopian point, which means 13.2% better optimized than Cielo-BASE's. Tables 20 and 21 demonstrate that LP successfully aids Cielo-LP to boost the optimality of its solution with reasonable computational costs.

Table 22 compares the objective values of $Cielo - LP_{WS}$ with the optimal results that LP finds with and without constraints. Here, LP is used to solve the entire problem. Since LP can consider only one objective in a single simulation run, Table 22 shows the optimal objective values in four objectives by running LP four times ("LP only" in Table 22). LP is also configured to use a weighted-sum function that aggregates four objective values. In this configuration, LP runs once to obtain four objective values ("LP only (WS)" in Table 22). Those values are not guaranteed to be optimal. With constraints disabled (C_∞), $Cielo - LP_{WS}$ and LP with a weighted-sum function produce higher/worse objective values than LP's optimal values because the two algorithms consider multiple objectives simultaneously. Note that $Cielo - LP_{WS}$'s performance is very close to LP in CPU and bandwidth allocation while it is inferior to LP in power consumption. $Cielo - LP_{WS}$ outperforms LP with a weighted-sum function in three of four objectives.

With constraints enabled (C_M), LP obtains the optimal objective values in CPU allocation and power consumption. However, it fails to obtain the optimal values in two other objectives within the timeout period of one hour. LP with a weighted-sum function fails to complete its execution within the timeout period. $Cielo - LP_{WS}$'s performance is very close to LP in CPU allocation while it is inferior to LP in power consumption.

Table 22 also shows the execution time for the three algorithms. LP's execution time indicates the total execution time to run four simulation runs. With constraints disabled, LP and LP with a weighted-sum function are significantly efficient compared to $Cielo - LP_{WS}$. With constraints enabled, their efficiency dramatically degrades. LP's execution time is five seconds and six seconds to obtain the objective values of CPU allocation and power consumption, respectively. Therefore, LP's execution time is greater than 7,211 seconds. The existence of constraints greatly impacts the execution time of LP and LP with a weighted-sum function while it does not impact $Cielo - LP_{WS}$. $Cielo - LP_{WS}$'s execution time increases only two seconds by enabling constraints. In summary,

$Cielo - LP_{WS}$ yields near LP-optimal performance in some objectives and it is robust against the existence of constraints.

Table 23 compares the convergence speed of $Cielo - LP_{WS}$ and Cielo-BASE based on the hypervolume measurement. $Cielo - LP_{WS}$ and Cielo-BASE spend 8 and 16 generations, respectively, to reach the hypervolume of 0.37. The speedup of $Cielo - LP_{WS}$ is 2.0. Cielo-BASE reaches the hypervolume of 0.3858 in 500 generations. In contrast, $Cielo - LP_{WS}$ requires only 91 generations to reach the same hypervolume, thereby yielding 5.5x speedup. These results illustrate that LP successfully aids Cielo-LP to boost its convergence speed.

Table 24 compares Cielo-LP with NSGA-II along with FFA and BFA based on their minimum, average and maximum objective values. Cielo-LP outperforms NSGA-II in all objectives except response time. Considering all four objectives, Cielo-LP yields 24.67% higher performance than NSGA-II. FFA and BFA produce two extreme results. FFA yields the lowest power consumption (59.61 Watts) because it is designed to place VMs on the minimum number of hosts; however, it sacrifices the other objectives. BFA performs the best in CPU allocation (28.28%) because it is designed to place VMs on the hosts that maintain higher resource availability. Cielo maintains balanced objective values in between FFA and BFA while performing better in response time, CPU allocation and bandwidth allocation.

Table 25 shows the variance of objective values that Cielo-LP and NSGA-II yield in 20 different simulation runs. A lower variance means higher stability (or higher similarity) in objective values (i.e., lower oscillations in objective values) among different simulation runs. Cielo-LP maintains significantly higher stability than NSGA-II in all objectives. Cielo-LP's average stability is 97.21% higher than NSGA-II's. This result exhibits Cielo's stability property (i.e. ability to seek evolutionarily stable strategies), which NSGA-II does not have.

Fig. 34 shows two three-dimensional objective spaces that plot a set of dominant strategies obtained from individual populations at each generation. Each blue dot indicates the average objective values that dominant strategies yield at a particular generation in 20 simulation runs. The trajectory of blue dots illustrates a path through which Cielo-LP's strategies evolve and improve objective val-

ues. Gray and red dots represent 20 different sets of objective values at the first and last generation in 20 simulation runs, respectively. While initial (gray) dots disperse (because strategies are generated at random initially), final (red) dots are overlapped in a small region. Consistent with Table 25, Fig. 34 verifies Cielo-LP's stability: reachability to at least one Nash equilibria regardless of the initial conditions.

4.6.4 Conclusion

This section proposes and evaluates Cielo-LP, an evolutionary game theoretic algorithm for adaptive and stable virtual machine (VM) placement in DVFS-enabled clouds. It theoretically guarantees that every application seeks an evolutionarily stable deployment strategy, which is an equilibrium solution under given workload and resource availability. Simulation results verify that Cielo-LP performs VM placement in an adaptive and stable manner. By integrating linear programming, Cielo-LP successfully gains performance improvement in optimality and convergence speed with reasonable computational costs. Cielo-LP outperforms existing well-known heuristics in the quality and stability of VM placement.

Applications successfully leverage DVFS to balance their response time performance, resource utilization and power consumption. Linear programming aids Cielo-LP to gain up to 38% improvement in optimality and up to 5.5x speedup in convergence speed with reasonably acceptable computational costs. Cielo-LP's performance is evaluated in comparison to existing heuristic algorithms. Cielo outperforms a well-known multiobjective evolutionary optimization algorithm, NSGA-II [DAP00] by 24% in optimality while maintaining 97% higher stability (lower oscillations). Cielo-LP also outperforms two other well-known heuristics, first-fit and best-fit algorithms (FFA and BFA), which have been widely used for adaptive cloud application deployment [LQL13, MLL12, GP13, CSV12].

Algorithm 4.4.1: Evolutionary Process in Cielo

$g = 0$
Randomly generate the initial N populations for N applications: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$
while $g < G_{max}$ **do**
  for each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
    $\mathscr{P}'_i \leftarrow \emptyset$
    for $j = 1$ to $|\mathscr{P}_i|/2$ **do**
      $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
      $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
      $winner \leftarrow$ performGame(s_1, s_2)
      $replica \leftarrow$ replicate($winner$)
      if random() $\leq P_m$ **then**
        $replica \leftarrow$ mutate($winner$)
      end if
      $\mathscr{P}_i \setminus \{s_1, s_2\}$
      $\mathscr{P}'_i \cup \{winner, replica\}$
    end for
    $\mathscr{P}_i \leftarrow \mathscr{P}'_i$
    $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
    while d_i is infeasible **do**
      $\mathscr{P}_i \setminus \{d_i\}$
      $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
    end while
    Deploy VMs for the current application based on d_i.
  end for
  $g = g + 1$
end while

Application type	1	2	3	4	5
Message arrival rate (λ in Eq. 4.6)	110	70	40	20	10
Web server (T_1^p in Eq. 4.4)	0.02	0.02	0.04	0.04	0.08
App server (T_2^p in Eq. 4.4)	0.03	0.08	0.04	0.13	0.11
DB server (T_3^p in Eq. 4.4)	0.05	0.05	0.12	0.08	0.11

Table 1: Message Arrival Rate and Message Processing Time

Table 2: Cielo Simulation Settings

Parameter	**Value**
Number of hosts (M)	100
Number of applications (N)	200
Number of simulation runs	20
Number of generations (G_{max})	500
Population size ($\|\mathscr{P}_i\|$)	100
Energy consumption for a single bit of data (e_t)	0.001 Watt
Upper limit of processing capacity per core(L_U)	100%
Upper limit of bandwidth capacity per host(L_E)	1Kbps
Upper limit of energy consumption per host(L_E)	400Watts
Upper limit of response time per application(L_R)	40ms

Table 3: P-states in Intel Core2 Quad Q6700

p-state	CPU frequency (f)	P_{idle}^{f}	P_{max}^{f}
p1	1.6 GHz	82.7 W	88.77 W
p2	1.867 GHz	82.85 W	92 W
p3	2.113 GHz	82.91 W	95.5 W
p4	2.4 GHz	83.1 W	99.45 W
p5	2.67 GHz	83.25 W	103 W

Table 4: Cielo Execution Time Comparison

Algorithms	Execution Time
Cielo-PD	6 min 48 sec
Cielo-HV	6 min 15 sec
Cielo-PD-HV	7 min 12 sec

Table 5: Performance of Cielo, FFA and BFA

Objectives		Min	Avg	Max
CPU allocation (%/host)	Cielo HV	7.97	8.04	8.12
	Cielo PD	19.38	19.69	19.93
	Cielo PD-HV	19.46	19.51	19.58
	FFA	96.1	96.1	96.1
	BFA	10.54	10.54	10.54
Bandwidth allocation (bps/host)	Cielo HV	199.86	200.95	202.44
	Cielo PD	224.54	228.39	232.32
	Cielo PD-HV	223.62	225.3	227.88
	FFA	445	446	446.92
	BFA	425	425	425
Power consumption (W)	Cielo HV	334.76	334.89	335
	Cielo PD	338.99	339.22	339.44
	Cielo PD-HV	339.08	339.19	339.34
	FFA	43.82	43.83	43.85
	BFA	338.66	338.73	338.8
Response time (msec)	Cielo HV	25.35	26.11	26.94
	Cielo PD	35.84	36	36.59
	Cielo PD-HV	30.64	30.93	31.33
	FFA	173.45	173.45	173.45
	BFA	152.92	152.92	152.92

Algorithm 4.5.1: Evolutionary Process in AGEGT

$g = 0$
Randomly generate N populations for N applications: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$
while $g < G_{max}$ **do**
  for each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
    $\mathscr{P}_i' \leftarrow \emptyset$
    for $j = 1$ to $|\mathscr{P}_i|/2$ **do**
      $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
      $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
      $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
      $replica \leftarrow$ replicate($winner$)
      $a_{i,t} \leftarrow argmax_{a_{i,t} \in a_i}\ u_t(replica)$
      for each parameter v in $a_{i,t}$ **do**
        if random() $\leq P_m$ **then**
          $replica \leftarrow$ mutate($replica$, v)
        end if
      end for
      $winner' \leftarrow$ performGame($loser$, $replica$)
      $\mathscr{P}_i \setminus \{s_1, s_2\}$
      $\mathscr{P}_i' \cup \{winner, winner'\}$
    end for
    $\mathscr{P}_i \leftarrow \mathscr{P}_i'$
    $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
    while d_i is infeasible **do**
      $\mathscr{P}_i \setminus \{d_i\}$
      $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
    end while
    Use d_i to adjust the parameters for VMs of the current application.
  end for
  $g = g + 1$
end while

Algorithm 4.5.2: Evolutionary Process in EGT-GLS

1: $g = 0$
2: Randomly generate N populations for N applications: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$
3: **while** $g < G_{max}$ **do**
4: **for** each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
5: $\mathscr{P}_i' \leftarrow \emptyset$
6: **for** $j = 1$ to $|\mathscr{P}_i|/2$ **do**
7: $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
8: $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
9: $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
10: $replica \leftarrow$ replicate($winner$)
11: **for** each parameter v in $replica$ **do**
12: **if** random() $\leq P_m$ **then**
13: $replica \leftarrow$ mutate($replica$, v)
14: **end if**
15: **end for**
16: $winner' \leftarrow$ performGame($loser$, $replica$)
17: $\mathscr{P}_i \setminus \{s_1, s_2\}$
18: $\mathscr{P}_i' \cup \{winner, winner'\}$
19: **end for**
20: $\mathscr{P}_i \leftarrow \mathscr{P}_i'$
21: $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
22: **while** d_i is infeasible **do**
23: $\mathscr{P}_i \setminus \{d_i\}$
24: $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
25: **end while**
26: $d_i \leftarrow$ localSearch(d_i)
27: Use d_i to adjust the parameters for VMs of the current application.
28: **end for**
29: $g = g + 1$
30: **end while**

Algorithm 4.5.3: Guided Local Search (localSearch())

Require: d_i: Dominant strategy to improve

Ensure: Improved dominant strategy

1: **for** $i = 1$ to Q **do**
2: **for** each t-th tier VM in d_i **do**
3: $RankedVM[] \leftarrow util_{t(CPU,BW,EN,RT)}(strategy)$
4: **end for**
5: $PenalizedVM \leftarrow RankedVM[3]$
6: **for** each parameter v in $PenalizedVM$ **do**
7: **if** random() $\leq P_m$ **then**
8: $replica \leftarrow$ mutate(d_i, v)
9: **end if**
10: **end for**
11: $d_i \leftarrow$ performGame($replica$, d_i)
12: **end for**
13: **return** d_i

Application type	1	2	3	4	5
Message arrival rate (λ in Eq. 4.6)	110	70	40	20	10
Web server (T_1^p in Eq. 4.4)	0.02	0.02	0.04	0.04	0.08
App server (T_2^p in Eq. 4.4)	0.03	0.08	0.04	0.13	0.11
DB server (T_3^p in Eq. 4.4)	0.05	0.05	0.12	0.08	0.11

Table 6: Message Arrival Rate and Message Processing Time

P-state	Frequency (q)	P^q_{idle}	P^q_{max}
p1	1.600 GHz	82.70 W	88.77 W
p2	1.867 GHz	82.85 W	92.00 W
p3	2.113 GHz	82.91 W	95.50 W
p4	2.400 GHz	83.10 W	99.45 W
p5	2.670 GHz	83.25 W	103.00 W

Table 7: P-states in Intel Core2 Quad Q6700

Parameter	Value
Number of hosts (M)	100
Number of cores per CPU/host (O in Eq. 4.6)	4
Number of applications (N)	200
Number of generations (G_{max} in Algo. 4.5.1)	500
Population size ($\|\mathscr{P}_i\|$ in Algo. 4.5.1)	100
Penalization rate (η in Algo. 4.5.1)	0.5
Mutation rate (P_m in Algo. 4.5.1)	$1/v$
Number of local search iterations (Q in Algo. 4.5.3)	20
Reference point for HV computation (x_r in Eq. 4.15)	f_C=400, f_B=4k, f_{PC}=40k, f_{RT}=400

Table 8: Parameter Settings for AGEGT

Constraint Combinations	C_C (%)	C_B (Kbps)	C_{PC} (W)	C_{RT} (ms)
C_∞	∞	∞	∞	∞
C_M	100	1,000	400	40

Table 9: Constraint Combinations

Distribution Index	HV	Distribution Index	HV
30	0.823	35	0.828
40	0.830	45	0.827
50	0.825		

Table 10: Impacts of Distribution Index Values on Hypervolume (HV) Performance in AGEGT

	HV=0.813	HV=0.830	HV=0.832	HV=0.835
AGEGT	6	45	66	323
EGT-GLS	5	123	370	—
EGT	17	213	—	—
NSGA-II	182	—	—	—

Table 11: Convergence Speed of AGEGT, EGT-GLS, EGT and NSGA-II

	AGEGT	NSGA-II (Avg)	NSGA-II (Closest)
Euclidean Distance	0.388	0.653	0.572
Manhattan Distance	0.618	1.186	0.983

Table 12: Comparison of AGEGT and NSGA-II with Distance Metrics

Objectives		Min	Avg	Max
CPU allocation (%/app)	AGEGT	15.00	15.05	15.12
	NSGA-II	28.86	30.29	31.35
	FFA	28.68	28.68	28.68
	BFA	28.94	28.94	28.94
Bandwidth allocation (Kbps/app)	AGEGT	238.55	238.65	238.71
	NSGA-II	278.32	288.27	295.89
	FFA	1186	1186	1186
	BFA	1200	1200	1200
Power consumption (W/app)	AGEGT	285.71	286.72	288.60
	NSGA-II	1245.15	1246	1246.92
	FFA	59.12	59.61	60.02
	BFA	341.74	341.85	341.95
Response time (msec/app)	AGEGT	22.77	22.78	22.80
	NSGA-II	11.56	11.79	12.04
	FFA	109.06	109.06	109.06
	BFA	92.09	92.09	92.09

Table 13: Comparison of AGEGT, NSGA-II, FFA and BFA in Objective Values

Objectives	AGEGT	NSGA-II	Diff (%)
CPU allocation	0.06	2.16	97.22%
Bandwidth allocation	0.001	5.599	99.98%
Power consumption	0.010	0.747	98.66%
Response time	0.001	0.239	99.58%
Average Difference (%)	–	–	98.86%

Table 14: Stability of Objective Values in AGEGT and NSGA-II

Algorithm 4.6.1: Evolutionary Process in Cielo-LP

Randomly generate the initial N populations for N applications: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$, and $g = 0$
while $g < G_{max}$ **do**
  for each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
    $\mathscr{P}'_i \leftarrow \emptyset$
    if random() $\leq P_l$ **then**
      $d_i \leftarrow$ linearProgramming($\mathscr{P}_i$)
    else
      for $j = 1$ to $|\mathscr{P}_i|/2$ **do**
        $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
        $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
        $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
        $replica \leftarrow$ replicate($winner$)
        for each parameter v in $replica$ **do**
          if random() $\leq P_m$ **then**
            $replica \leftarrow$ mutate($replica$, v)
          end if
        end for
        $winner' \leftarrow$ performGame($loser$, $replica$)
        $\mathscr{P}_i \setminus \{s_1, s_2\}$
        $\mathscr{P}'_i \cup \{winner, winner'\}$
      end for
      $\mathscr{P}_i \leftarrow \mathscr{P}'_i$
      $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
      while d_i is infeasible **do**
        $\mathscr{P}_i \setminus \{d_i\}$
        $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
      end while
      Deploy VMs for the current application based on d_i.
    end if
  end for
  $g = g + 1$
end while

Application type	1	2	3	4	5
Message arrival rate (λ in Eq. 4.6)	110	70	40	20	10
Web server (T_1^p in Eq. 4.4)	0.02	0.02	0.04	0.04	0.08
App server (T_2^p in Eq. 4.4)	0.03	0.08	0.04	0.13	0.11
DB server (T_3^p in Eq. 4.4)	0.05	0.05	0.12	0.08	0.11

Table 15: Message Arrival Rate and Message Processing Time

P-state	Frequency (q)	P_{idle}^q	P_{max}^q
p1	1.600 GHz	82.70 W	88.77 W
p2	1.867 GHz	82.85 W	92.00 W
p3	2.113 GHz	82.91 W	95.50 W
p4	2.400 GHz	83.10 W	99.45 W
p5	2.670 GHz	83.25 W	103.00 W

Table 16: P-states in Intel Core2 Quad Q6700

Parameter	Value
Number of hosts (M)	100
Number of CPU cores per host (O in Eq. 4.6)	4
Number of applications (N)	200
Number of generations (G_{max} in Algo. 4.6.1)	500
Population size ($\|\mathscr{P}_i\|$ in Algo. 4.6.1)	100
Linear programming rate (P_l in Algo. 4.6.1)	0.005, 0.05, 0.1, 0.5
Mutation rate (P_m in Algo. 4.6.1)	$1/v$
Reference point for HV computation	f_C=600, f_B=1000, f_{PC}=2000, f_{RT}=1000

Table 17: Parameter Settings for Cielo-LP

Constraint Combinations	C_C (%)	C_B (Kbps)	C_{PC} (W)	C_{RT} (ms)
C_∞	∞	∞	∞	∞
C_M	100	1,000	400	40

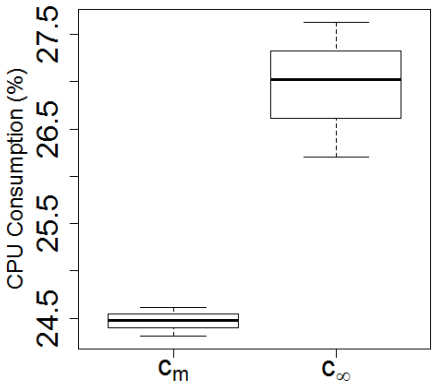
Table 18: Constraint Combinations

Distribution Index	HV	Distribution Index	HV
30	0.823	35	0.828
40	0.830	45	0.827
50	0.825		

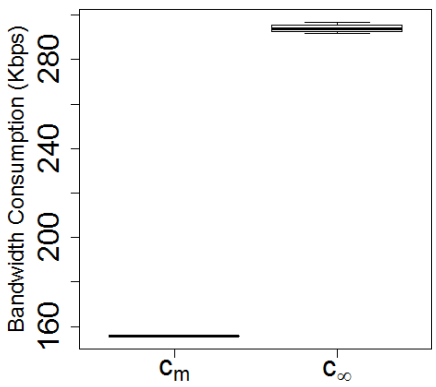
Table 19: Impacts of Distribution Index Values on Hypervolume (HV) Performance in Cielo-LP

Algorithms	LP rate (%)	Execution time (s)	Speedup
Cielo-BASE	0%	393	1.0
$Cielo-LP_{WS}$ w/ f_{RT}^{LP}	0.5%	>3600	<0.10
	5%	>3600	<0.10
	10%	>3600	<0.10
	50%	>3600	<0.10
$Cielo-LP_{WS}$ w/o f_{RT}^{LP}	0.5%	435	0.90
	5%	556	0.70
	10%	832	0.47
	50%	>3600	<0.10

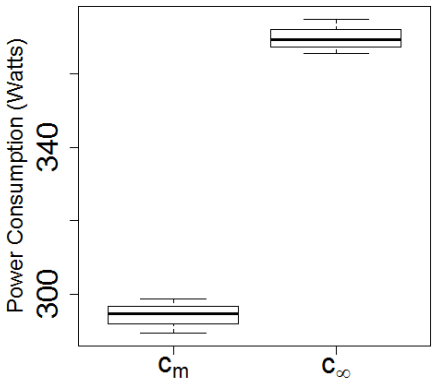
Table 20: Impacts of LP Rates on the Execution Time Performance



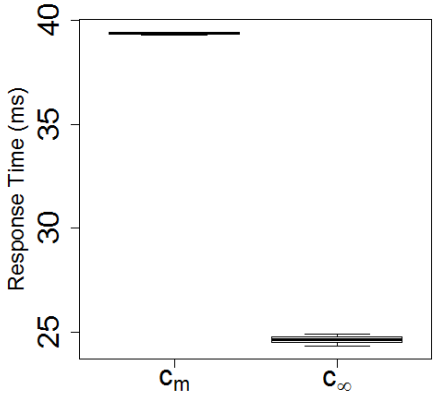
(a) f_C



(b) f_B

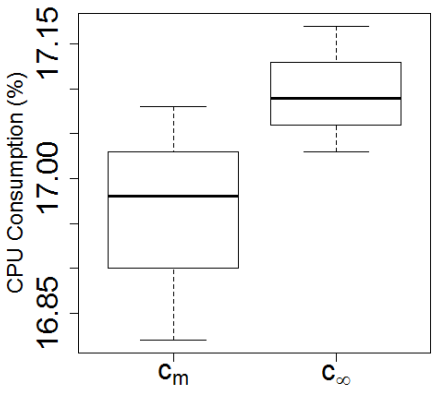


(c) f_{PC}

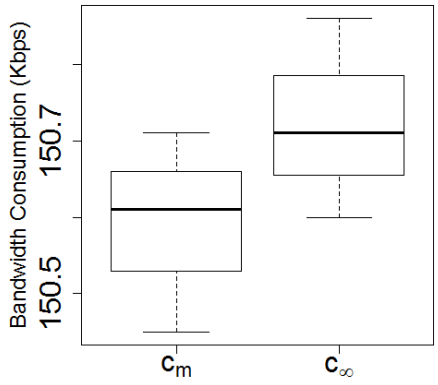


(d) f_{RT}

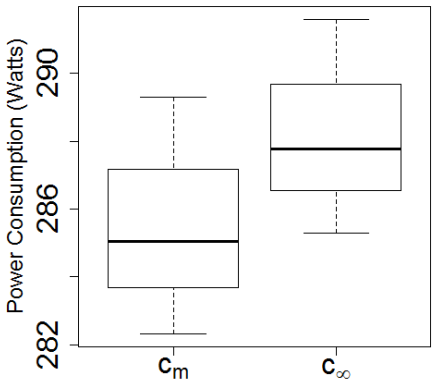
Figure 30: Cielo-BASE's Objective Values with and without Constraints (C_M and C_∞



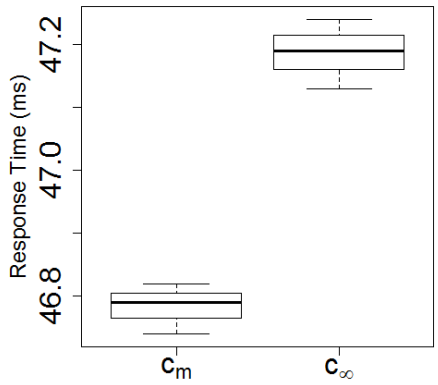
(a) f_C



(b) f_B

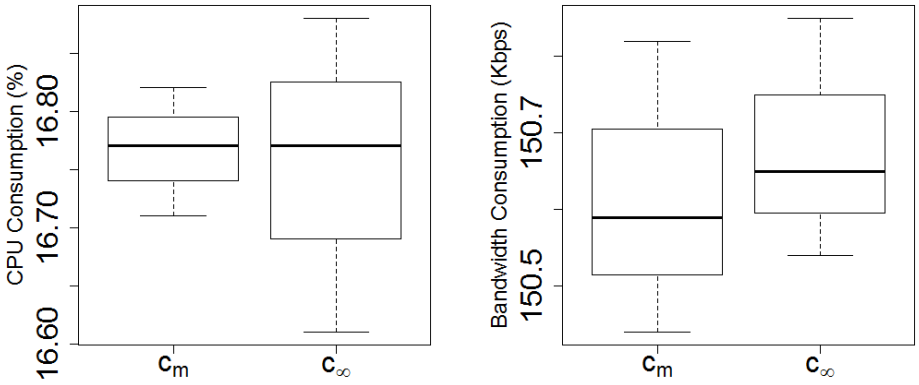


(c) f_{PC}



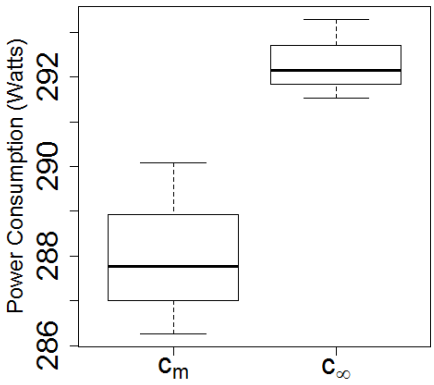
(d) f_{RT}

Figure 31: $Cielo - LP_{WS}$'s Objective Values w/ & w/o Constraints (C_M and C_∞). LP rate: 0.5%

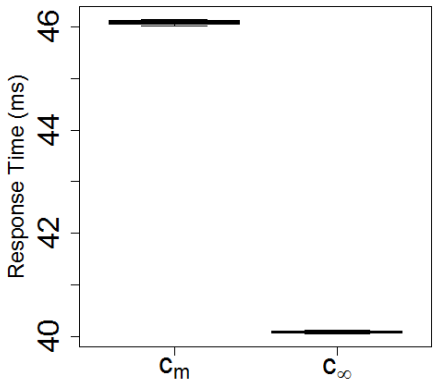


(a) f_C

(b) f_B

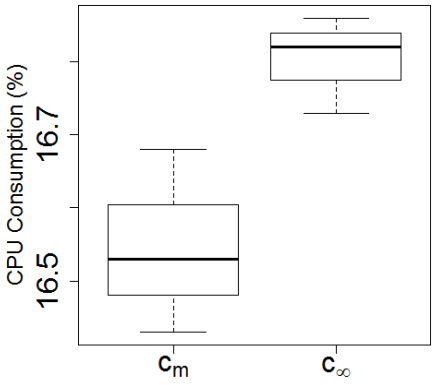


(c) f_{PC}

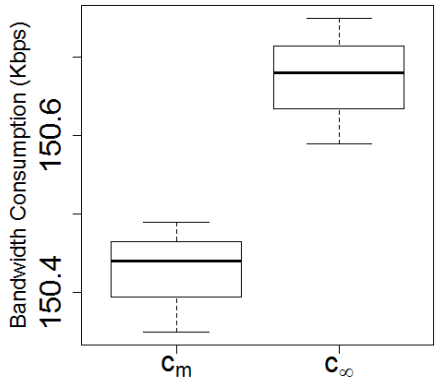


(d) f_{RT}

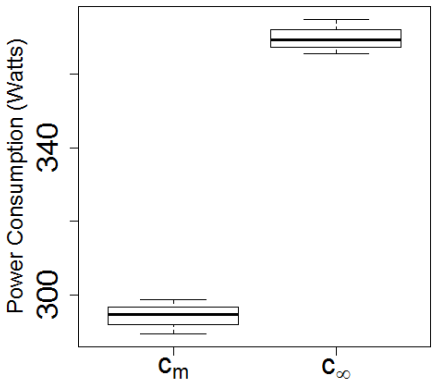
Figure 32: $Cielo - LP_{WS}$'s Objective Values w/ & w/o Constraints (C_M and C_∞). LP rate: 5%



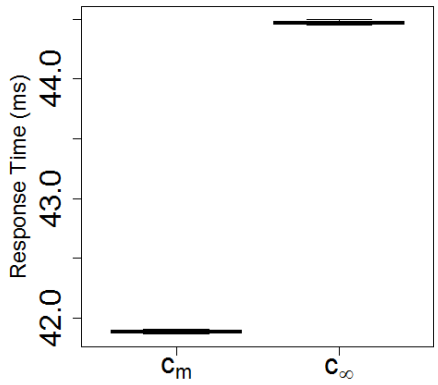
(a) f_C



(b) f_B



(c) f_{PC}



(d) f_{RT}

Figure 33: $Cielo - LP_{WS}$'s Objective Values w/ & w/o Constraints (C_M and C_∞). LP rate: 10%

	LP_C	LP_{BW}	LP_{PC}	LP_{WS}	BASE
f_C (%)	15.04 (38.6%)	23.03	18.13	16.53 (32.5%)	24.48
f_B (Kbps)	155.87	150.12 (3.6%)	151.43	150.44 (3.4%)	155.76
f_{PC} (W)	296.98	294.78	274.23 (6.9%)	283.25 (3.8%)	294.54
f_{RT} (ms)	48.76	47.54	46.30	41.89	39.39
Distance	0.510 (12.8%)	0.574 (1.9%)	0.515 (12.0%)	0.508 (13.2%)	0.585

Table 21: Performance Improvement of Cielo-LPs against Cielo-BASE

	f_C	f_B	f_{PC}	f_{RT}	Exec. time (s)
$Cielo - LP_{WS}$: C_∞	16.82	150.68	290.33	44.47	832
LP only: C_∞	15.00	150	54.56	8.06	23
LP only (WS): C_∞	41.10	450	499.97	8.12	7
$Cielo - LP_{WS}$: C_M	16.53	150.44	283.25	41.89	834
LP only: C_M	15.00	—	54.56	—	>7,211
LP only (WS): C_M	—	—	—	—	>3,600

Table 22: Comparison of Objective Values and Execution Time between $Cielo - LP_{WS}$ and Linear Programming

	Hypervolume				
	0.60	0.70	0.75	0.81	0.83
Cielo-BASE	16	41	100	361	500
$Cielo - LP_{WS}$	8	14	37	79	91
Speedup	2.0	2.9	2.7	4.6	5.5

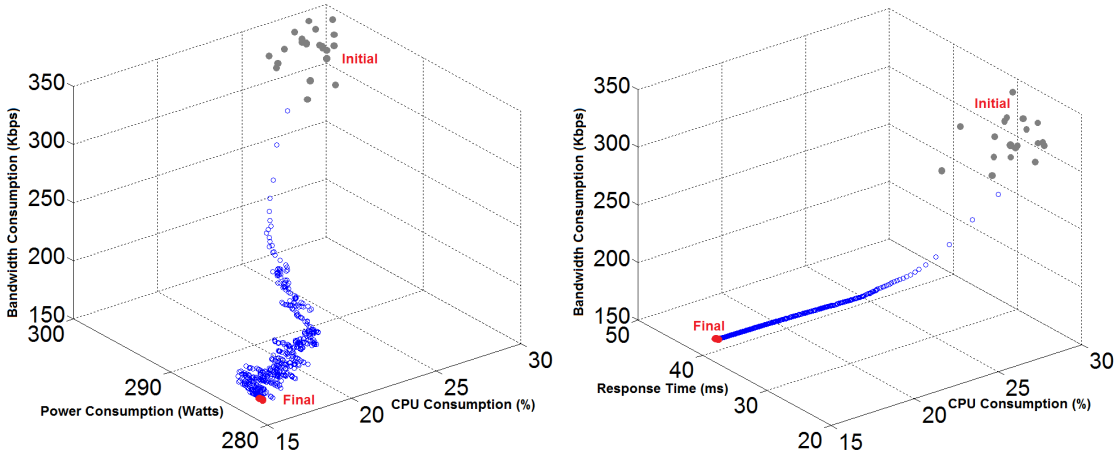
Table 23: Comparison of Convergence Speed between Cielo-BASE and $Cielo - LP_{WS}$

Objectives		Minimum	Average	Maximum
CPU allocation (%/app)	$Cielo - LP_{WS}$	16.43	16.53	16.68
	NSGA-II	28.86	30.29	31.35
	FFA	28.68	28.68	28.68
	BFA	28.28	28.28	28.28
Bandwidth allocation (Kbps/app)	$Cielo - LP_{WS}$	150.35	150.44	150.49
	NSGA-II	278.32	288.27	295.89
	FFA	1186	1186	1186
	BFA	1200	1200	1200
Power consumption (W/app)	$Cielo - LP_{WS}$	274.19	283.25	290.69
	NSGA-II	1245.15	1246	1246.92
	FFA	59.12	59.61	60.02
	BFA	341.74	341.85	341.95
Response time (msec/app)	$Cielo - LP_{WS}$	41.87	41.89	41.91
	NSGA-II	11.56	11.79	12.04
	FFA	109.06	109.06	109.06
	BFA	92.09	92.09	92.09

Table 24: Comparison of Objective Values among $Cielo - LP_{WS}$, NSGA-II, FFA and BFA

	$Cielo - LP_{WS}$	NSGA-II	Difference (%)
CPU allocation	0.150	2.160	93.05%
Bandwidth allocation	0.060	5.599	98.92%
Power consumption	0.005	0.747	99.40%
Response time	0.006	0.239	97.48%
Average	0.055	2.186	97.21%

Table 25: Stability of Objective Values in $Cielo - LP_{WS}$ and NSGA-II



(a) CPU allocation, Bandwidth allocation and Energy consumption

(b) CPU allocation, Bandwidth allocation and Response time

Figure 34: Trajectory of Cielo-LP's Solution through Generations

CHAPTER 5

BODY SENSOR NETWORK

5.1 Introduction

This Chapter considers a multi-tier architecture for cloud-integrated body sensor networks (BSNs), called Body-in-the-Cloud (BitC), which is intended to support home healthcare with on-body energy harvesting devices (e.g., piezoelectric and thermoelectric generators) as well as on-body physiological and activity monitoring sensors. It formulates a configuration problem in BitC and approaches the problem with an evolutionary game theoretic algorithm to configure BSNs in an adaptive and stable manner. BitC allows BSNs to adapt their configurations (i.e., sensing intervals and sampling rates as well as data transmission intervals) to operational conditions (e.g., data request patterns) with respect to multiple conflicting performance objectives such as resource consumption and data yield. In BitC, evolutionary multiobjective games are performed on configuration strategies (i.e., solution candidates) with an aid of local search mechanisms. BitC theoretically guarantees that each BSN performs an evolutionarily stable configuration strategy, which is an equilibrium solution under given operational conditions. Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies. This chapter evaluates five algorithmic variants of BitC under various settings and demonstrates that BitC allows BSNs to successfully leverage harvested energy to balance their performance in different objectives such as resource consumption and data yield.

5.2 State of the art

Various architectures and research tools have been proposed for cloud-integrated sensor networks including BSNs [DLO14, HSH09, AHS07, GWM04, BS10, SQM12, ACC10, BBS12, AG11, ZYS13, YK10, RKW10, PKG08, CYH13, FPP14]. Many of them, [DLO14, HSH09, AHS07, GWM04, BS10, SQM12, ACC10, BBS12, AG11, ZYS13], assume three-tier architectures similar to BitC and investigate publish/subscribe communication between the edge layer to the cloud layer. Their focus is placed on push communication. In contrast, BitC investigates push-pull hybrid communication between the sensor layer and the cloud layer through the edge layer. Yuriyama et al. [YK10], Rollin et al. [RKW10] and Chung et al. [CYH13] propose a two-tier architecture that consists of the sensor and cloud layers. The architectures proposed by Yuriyama et al. and Fortino et al. [FPP14] are similar to BitC in that they leverage the notion of virtual sensors. However, they do not consider push-pull (nor publish/subscribe) communication. All the above-mentioned relevant work do not consider adaptive/stable configurations of sensor networks as BitC does [HSH09, AHS07, GWM04, BS10, SQM12, ACC10, BBS12, AG11, ZYS13, YK10, RKW10, PKG08, CYH13, FPP14].

Push-pull hybrid communication has been studied in sensor networks [WBS10, BHS07, KK06, LGS06]. However, few efforts exist to study it between the edge and cloud layers in the context of cloud-integrated sensor networks. Unlike those relevant work, this paper formulates a sensor network configuration problem with cloud-specific objectives as well as the ones in sensor networks and seeks adaptive/stable solutions for the problem.

Xu et al. propose a three-tier architecture called CEB (Cloud, Edge and Beneath), which is similar to BitC, and investigate a mechanism to adapt data transmission rates between layers according to a given pattern of data requests [XHT11]. CEB runs two optimization algorithms collaboratively: OPT-1 and OPT-2, which optimize data transmission rates between the cloud and edge layers and between the edge and sensor layers, respectively. Optimization is carried out on a sensor node by sensor node basis with respect to a single objective: energy consumption. In contrast, BitC considers sensing intervals and sampling rates for sensors as well as data transmission rates for nodes

and runs a single algorithm for the entire group of sensor and sink nodes with respect to multiple conflicting objectives including energy consumption.

Kumrai [KOD14] proposes a novel incentive mechanism for participatory sensing based on the evolutionary algorithm. It considers energy consumption optimization problem similar to BitC, however it does not consider multi objective performance scenario.

SC-iPaaS (Sensor-Cloud Integration Platform as a Service) is similar to BitC in that both consider three-tier architecture for cloud-integrated BSNs and caries out a single algorithm for the entire group of sensor and sink nodes with respect to multiple conflicting objectives including energy consumption. SC-iPaaS uses an evolutionary game theoretic algorithm that retains stability (i.e. reachability to at least one Nash equilibria) as well as optimality in configuring BSNs while a genetic algorithm is used in SC-iPaaS [PSO14]. As stochastic global search algorithms, genetic algorithms lack stability.

5.3 Problem Formulation

BitC consists of the following three layers (Fig. 35).

Sensor Layer: operates one or more BSNs on a per-patient basis (Fig. 35). Each BSN contains one or more sensor node in a certain topology (e.g., tree, star or mesh topology). It assumes the star topology. Each sensor node is equipped with a sensor(s) and an energy harvester(s). It is assumed to be battery-operated. It supplies a limited amount of energy to a sensor(s) attached to it and receives power supply from an attached energy harvester(s) as it/they harvest energy. It maintains a sensing interval and a sampling rate for each attached sensor. Upon a sensor reading, it stores collected data in its own memory space. Given a data transmission interval, it periodically flushes all data stored in its memory space and transmits the data to a sink node.

Edge Layer: consists of sink nodes, each of which participates in a certain BSN and receives sensor data periodically from sensor nodes in the BSN. A sink node stores incoming sensor data in its memory space and periodically flushes stored data to transmit (or push) them to the cloud layer.

It maintains the mappings between physical and virtual sensors. In other words, it knows the origins and destinations of sensor data. Different sink nodes have different data transmission intervals. A sink node's data transmission interval can be different from the ones of sensor nodes in the same BSN. Sink nodes are assumed to have limited energy supplies through batteries.

In addition to pushing sensor data to a virtual sensor, each sink node receives a pull request from a virtual sensor when the virtual sensor does not haves sensor data that a cloud application(s) requires. If the sink node has the requested data in its memory, it returns that data. Otherwise, it issues another pull request to a sensor node that is responsible for the requested data. Upon receiving a pull request, the sensor node returns the requested data if it has the data in its memory. Otherwise, it returns an error message to a could application.

Cloud Layer: operates on clouds to host applications that allow medical staff to place sensor data requests on virtual sensors in order to monitor patients. If a virtual sensor has data that an application requests, it returns that data. Otherwise, it issues a pull request to a sink node. While push communication carries out a one-way upstream travel of sensor data, pull communication incurs a round trip for requesting data and receiving that data (or an error message).

Virtual sensors are java nodes running in the server on the cloud layer. These virtual sensor nodes are predefined by medical doctors. Each sensor is associated with a four digit identification number, the first two digit indicate the index of the BSN and the last two digit indicate the index of the sensor. Once data are pushed to the cloud layer, corresponding virtual sensor node responds to the arrived data by checking the identification number. Virtual sensors call storeToDB java method which issues a sql query that store the received data to the corresponding table in the database. Every time a request come in virtual sensors first check whether the data is available by calling isAvailable java method. If the data is available then it retrieves the data by calling getFromDB java method that issues a sql query to the corresponding table in the database, if not virtual sensors issues a pull request to Edge layer by calling pullData java method. Once virtual sensors get the desired data, it backs to user.

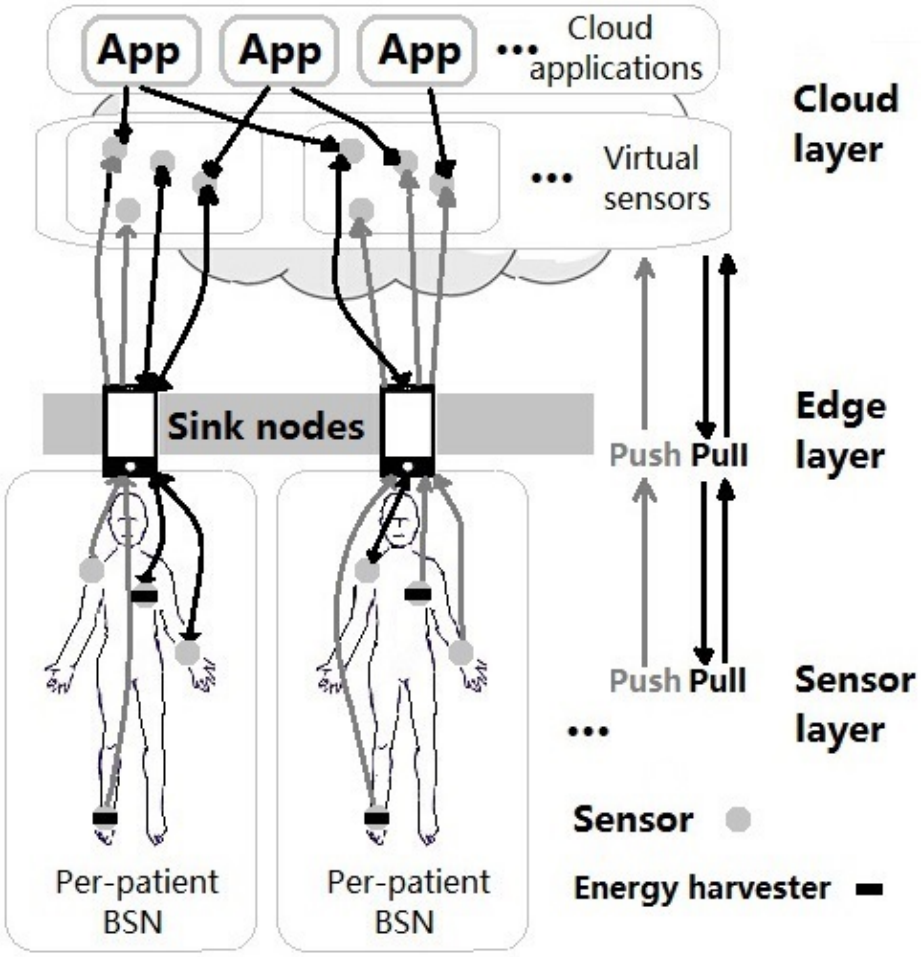


Figure 35: A Push-Pull Hybrid Communication in BitC

This section describes a BSN configuration problem for which BitC seeks equilibrium solutions. Each BSN configuration consists of four types of parameters (i.e., decision variables): sensing intervals and sampling rates for sensor nodes as well as data transmission intervals for sensor and sink nodes. The problem is stated with the following symbols.

- $B = \{b_1, b_2, ..., b_i, ..., b_N\}$ denotes the set of N BSNs, each of which operates for a patient.
- Each BSN b_i consists of a sink node (denoted by m_i) and M sensors: $b_i = \{s_{i1}, s_{i2}, ..., s_{ij}, ..., s_{iM}\}$. o_{ij} is the data transmission interval for s_{ij} to transmit sensor collected data. p_{ij} and q_{ij} are the

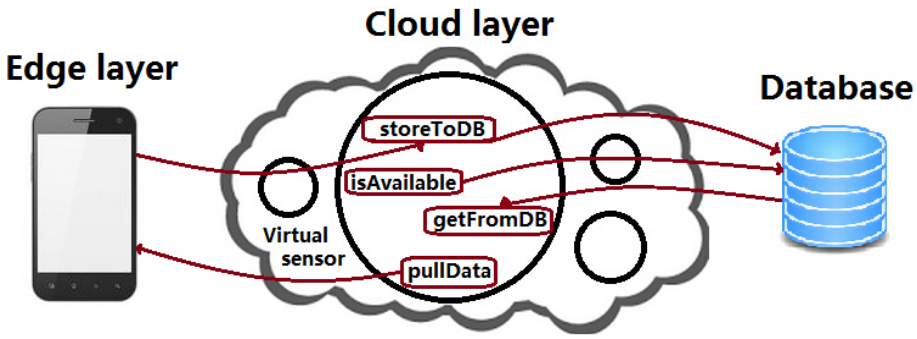


Figure 36: Virtual sensor communication diagram

sensing interval and sampling rate for s_{ij}. Sampling rate is defined as the number of sensor data samples collected in a unit time. Each sensor stores collected sensor data in its memory space until its next push transmission. If the memory becomes full, it performs FIFO (First-In-First-Out) data replacement. In a push transmission, it flushes and sends out all data stored in its memory.

- o_{m_i} denotes the data transmission interval for m_i to forward (or push) sensor data incoming from sensor nodes in b_i In between two push transmissions, m_i stores sensor data from b_i in its memory. It performs FIFO data replacement if the memory becomes full. In a push transmission, it flushes and sends out all data stored in the memory.

- $R_{ij} = \{r_{ij1}, r_{ij2}, ..., r_{ijr}, ..., r_{ij|R_{ijk}|}\}$ denotes the set of sensor data requests that cloud applications issue to the virtual counterpart of s_{ij} (s'_{ij}) during the time period of W in the past. Each request r_{ijr} is characterized by its time stamp (t_{ijr}) and time window (w_{ijr}). It retrieves all sensor data available in the time interval $[t_{ijr} - w_{ijr}, t_{ijr}]$. If s'_{ij} has at least one data in the interval, it returns those data. Otherwise, it issues a pull request to m_i.

- $R^m_{ij} \in R_{ij}$ denotes the set of sensor data requests for which a virtual sensor s'_{ij} has no data. $|R^m_{ij}|$ indicates the number of pull requests that s'_{ij} issues to m_i. In other words, $R_{ij} \setminus R^m_{ij}$ is the set of sensor data requests that s'_{ij} fulfills regarding s_{ij}.

- $R^s_{ij} \in R^m_{ij} \in R_{ij}$ denotes the set of sensor data requests for which m_i has no data. $|R^s_{ij}|$ indicates the number of pull requests that m_i issues to h_{ij} for collecting data from s_{ij}. $R^m_{ij} \setminus R^s_{ij}$ is the set of sensor data requests that m_i fulfills regarding s_{ij}.

This study considers four performance objectives: bandwidth consumption between the edge and cloud layers (f_B), energy consumption of sensor and sink nodes (f_E), request fulfillment for cloud applications (f_R) and data yield for cloud applications (f_D). The first two objectives are to be minimized while the others are to be maximized.

The bandwidth consumption objective (f_B) is defined as the total amount of data transmitted per a unit time between the edge and cloud layers. This objective impacts the payment for bandwidth consumption based on a cloud operator's pay-per-use billing scheme. It also impacts the lifetime of sink nodes. f_B is computed as follows.

$$\begin{aligned} f_B = & \frac{1}{W} \sum_{i=1}^{N} \sum_{j=1}^{M} (c_{ij} d_{ij}) + \frac{1}{W} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{r=1}^{|R^m_{ij}|} (\phi_{ijr} d_{ij} + d_r) \\ & + \frac{1}{W} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{r=1}^{|R^s_{ij}|} e_r (|R^s_{ij}| - \eta_{ijr}) \end{aligned} \tag{5.1}$$

The first and second terms indicate the bandwidth consumption by one-way push communication from the edge layer to the cloud layer and two-way pull communication between the cloud and edge layers, respectively. c_{ij} denotes the number of sensor data that s_{ij} generates and sink nodes in turn push to the cloud layer during W. d_{ij} denotes the size of each sensor data (in bits) that s_{ij} generates. It is currently computed as: $q_{ij} \times 16$ bits/sample. ϕ_{ijr} denotes the number of sensor data that a pull request $r \in R^m_{ij}$ can collect from sink nodes ($\phi_{ijr} = |R^m_{ij} \setminus R^s_{ij}|$). d_r is the size of a pull request transmitted from the cloud layer to the edge layer. The third term in Eq. 5.1 indicates the bandwidth consumption by the error messages that sensors generate because they fail to fulfill pull requests. η_{ijr} is the number of sensor data that a pull request $r \in R^s_{ij}$ can collect from sensor nodes. e_r is the size of an error message.

The energy consumption objective (f_E) is defined as the total amount of energy that sensor and sink nodes consume for data transmissions during W. It impacts the lifetime of sensor and sink nodes. It is computed as follows.

$$\begin{aligned} f_E = & \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{W}{o_{ij}}e_t d_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{r=1}^{|R_{ij}^s|} e_t \eta_{ijr}(d_{ij}+d'_r) \\ & + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\frac{W}{o_{m_i}}e_t d_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{r=1}^{|R_{ij}^m|} e_t \phi_{ijr}(d_{ij}+d_r) \\ & + 2\times\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{r=1}^{|R_{ij}^s|} e_t e_r(|R_{ij}^s| - \eta_{ijr}) \end{aligned} \tag{5.2}$$

The first and second terms indicate the energy consumption by one-way push communication from the sensor layer to the edge layer and two-way pull communication between the edge layer and the sensor layer, respectively. e_t denotes the amount of energy (in Watts) that a sensor or sink node consumes to transmit a single bit of data. d'_r denotes the size of a pull request from the edge layer to the sensor layer. The third and fourth terms indicate the energy consumption by push and pull communication between the edge and cloud layer, respectively. The fifth term indicates the energy consumption for transmitting error messages on sensor and sink nodes.

The request fulfillment objective (f_R) is the ratio of the number of fulfilled requests over the total number of requests:

$$f_R = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{r=1}^{|R_{ij}|} I_{R_{ij}}}{|R_{ij}|} \times 100 \tag{5.3}$$

$I_{R_{ij}} = 1$ if a request $r \in R_{ij}$ obtains at least one sensor data; otherwise, $I_{R_{ij}} = 0$.

The data yield objective (f_Y) is defined as the total amount of data that cloud applications gather for their users. This objective impacts the informedness and situation awareness for application users. It is computed as follows.

$$f_Y = \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{r=1}^{|R_{ij}^m|} \phi_{ijr} + \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{r=1}^{|R_{ij}^s|} \eta_{ijr} + c_{ij} \tag{5.4}$$

BitC considers four constraints.

- The first constraint (C_E) is the upper limit for energy consumption: $f_E < C_E$. A violation for the constraint (g_E) is computed as

  $$g_E = I_E \times (f_E - C_E) \tag{5.5}$$

  where $I_E = 1$ if $f_E > C_E$; otherwise $I_E = 0$.

- The second constraint (C_Y) is the lower limit for data yield: $f_Y > C_Y$. A constraint violation (g_Y) is computed as

  $$g_Y = I_Y \times (C_Y - f_Y) \tag{5.6}$$

  where $I_Y = 1$ if $f_Y < C_Y$; otherwise $I_Y = 0$.

- The third constraint (C_R) is the lower limit for request fulfillment: $f_R > C_R$. A constraint violation in request fulfillment (g_R) is computed as

  $$g_R = I_R \times (C_R - f_R) \tag{5.7}$$

  where $I_R = 1$ if $f_R < C_R$; otherwise $I_R = 0$.

- The fourth constraint (C_B) is the upper limit for bandwidth consumption: $f_B < C_B$. A violation for this constraint (g_B) is computed as

  $$g_B = I_B \times (f_B - C_B) \tag{5.8}$$

  where $I_B = 1$ if $f_B > C_B$; otherwise $I_B = 0$.

5.4 BitC

Body sensor networks (BSNs) are expected to aid pervasive healthcare with on-body sensors by remotely and continuously performing physiological and activity monitoring for patients [CGV11, PPB12]. Body-in-the-Cloud (BitC) virtualizes per-patient BSNs onto clouds by taking advantage of cloud computing features such as pay-per-use billing, scalability in data storage and processing, availability through multi-regional application deployment and accessibility through universal communication protocols (e.g., HTTP and REST). BitC assumes energy harvesting aware BSNs, each of which operates on-body energy harvesting devices (e.g., piezoelectric and thermoelectric generators) as well as on-body sensors for, for example, heart rate, oxygen saturation, body temperature and fall detection.

BitC consists of the sensor, edge and cloud layers (Fig. 35). The sensor layer is a collection of sensor nodes in BSNs. Each BSN operates one or more sensor nodes, each of which is equipped with a sensor(s) and an energy harvester(s). Sensor nodes are wirelessly connected to a dedicated per-patient device or a patient's computer (e.g., smartphone or tablet machine) that serves as a sink node. The edge layer consists of sink nodes, which collect sensor data from sensor nodes in BSNs. The cloud layer consists of cloud environments that host virtual sensors, which are virtualized counterparts (or software counterparts) of physical sensors in BSNs. Virtual sensors collect sensor data from sink nodes in the edge layer and store those data for future use. The cloud layer also hosts various applications that obtain sensor data from virtual sensors and aid medical staff (e.g., clinicians, hospital/visiting nurses and caregivers) to monitor patients and share sensor data for clinical observation and intervention.

BitC performs push-pull hybrid communication between its three layers. Each sensor node periodically collects data from a sensor(s) attached to it based on sensor-specific sensing intervals and sampling rates and transmits (or pushes) those collected data to a sink node. The sink node in turn forwards (or pushes) incoming sensor data periodically to virtual sensors in clouds. When a virtual sensor does not have sensor data that a cloud application requires, it obtains (or pulls) that

data from a sink node or a sensor node. This push-pull communication is intended to make as much sensor data as possible available for cloud applications by taking advantage of push communication while allowing virtual sensors to pull any missing or extra data anytime in an on-demand manner. For example, when an anomaly is found in pushed sensor data, medical staff may pull extra data in a higher temporal resolution to better understand a patient's medical condition. Given a sufficient amount of data, they may perform clinical intervention, order clinical cares, dispatch ambulances or notify family members of patients.

This study focuses on configuring BSNs in BitC by adjusting four types of parameters (i.e., sensing intervals and sampling rates for sensors as well as data transmission intervals for sensor and sink nodes) and studies two properties in configuring BSNs:

- **Optimality**: Seeking for the optimal BSN configurations according to operational conditions (e.g., data request patterns placed by cloud applications and availability of resources such as bandwidth and memory) with respect to performance objectives such as bandwidth consumption, energy consumption and data yield.
- **Stability**: Minimizing oscillations (non-deterministic inconsistencies) in making adaptation decisions. BitC considers stability as the reachability to at least one of equilibrium solutions in decision making. A lack of stability results in making inconsistent adaptation decisions in different attempts/trials with the same problem settings.

BitC leverages an evolutionary game theoretic algorithm to configure BSNs in an adaptive and stable manner. This chapter describes the design of BitC and evaluates its optimality and stability. In BitC, each BSN maintains a set (or a population) of configuration strategies (solution candidates), each of which specifies a set of configuration parameters for that BSN. BitC theoretically guarantees that, through a series of evolutionary games between BSN configuration strategies, the population state (i.e., the distribution of strategies) converges to an evolutionarily stable equilibrium, which is always converged to regardless of the initial state. (A dominant strategy in the evolutionarily stable population state is called an evolutionarily stable strategy (ESS).) In this state, no other strategies

except an ESS can dominate the population. Given this theoretical property, BitC allows each BSN to operate at equilibrium by using an ESS in a deterministic (i.e., stable) manner.

5.4.1 Algorithm

BitC maintains N populations, $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$, for N BSNs and performs games among strategies in each population. Each strategy $s(b_i)$ specifies a particular configuration for a BSN b_i using four types of parameters: sensing intervals and sampling rates for sensors (p_{ij} and q_{ij}) as well as data transmission intervals for sink and sensor nodes (o_{m_i} and o_{ij}).

$$s(b_i) = \bigcup_{j \in 1..M} (o_{m_i}, o_{ij}, p_{ij}, q_{ij}) \;\; 1 < i < N \tag{5.9}$$

Algorithm 5.4.1 shows how BitC seeks an evolutionarily stable configuration strategy for each BSN through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$ (Line 2). Those strategies may or may not be feasible. Note that a strategy is said to be feasible if it violates none of four constraints described in Section 5.3.

In each generation (g), a series of games are carried out on every population (Lines 4 to 28). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to performance objectives described in Section 5.3 (Lines 7 to 9). The winner is replicated to increase its population share and mutated with polynomial mutation (Lines 10 to 18) [DPA02]. Mutation randomly chooses a parameter (or parameters) in a given strategy with a certain mutation rate P_m and alters its/their value(s) at random (Lines 12 to 14). Then a game is performed between loser and the mutated winner (Line 16). Elitism concept is applied here to select the best two among strategies (winner, loser and mutated winner), and the worst strategy disappears in the population.

Once all strategies play games in the population, BitC identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 20 to 24). After a dominant strategy is determined, BitC performs local search to improve the dominant strategy

(Line 26). In the end, BitC configures a BSN with the parameters contained in the dominant strategy (Line 27).

A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (c.f. performGame() in Algorithm 5.4.1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins over its opponent. If both strategies are feasible, they are compared with one of the following five schemes to select the winner.

- Pareto dominance (PD): This scheme is based on the notion of dominance [SD95], in which a strategy s_1 is said to dominate another strategy s_2 if both of the following conditions hold:
  - s_1's objective values are superior than, or equal to, s_2's in all objectives.
  - s_1's objective values are superior than s_2's in at least one objectives.

  The dominating strategy wins a game over the dominated one. If two strategies are non-dominated with each other, the winner is randomly selected.

- Hypervolume (HV): This scheme is based on the hypervolume (HV) metric [ZT98]. It measures the volume that a given strategy s dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{5.10}$$

  Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

- Hybrid of Pareto dominance and hypervolume (PD-HV): This scheme is a combination of the above two schemes. First, it performs the Pareto dominance (PD) comparison for given two strategies. If they are non-dominated with each other, the hypervolume (HV) comparison is used to select the winner. If they still tie with the hypervolume metric, the winner is randomly selected.

- Maxmin (MM): This scheme is based on the maxmin (MM) metric [Coe98]. It measures how distant (i.e., better) a given strategy s is from the other strategies in a population ($s^{'} \in \mathscr{P}_i$).

$$MM(s) = \max_{s' \in \mathscr{P}_i \setminus \{s\}} \left\{ \min_k \left(s_k, s_k^{'} \right) \right\} \tag{5.11}$$

  s_k denotes the k-th objective value of the strategy s. Given two strategies, the one with a higher maxmin value wins a game. If both have the same maxmin value, the winner is randomly selected.

- Hybrid of Pareto dominance and maxmin (PD-MM): This scheme is a combination of the PD and MM schemes. First, it performs the Pareto dominance (PD) comparison for given two strategies. If they are non-dominated with each other, the MM comparison is used to select the winner. If they still tie with the maxmin metric, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. An infeasible strategy s_1 wins a game over another infeasible strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

5.4.1.1 Local Search

Local Search is an approach to improve the performance of a local dominant strategy. It attempts to search for a better quality dominant strategy by performing mutation on the current local dominant strategy. BitC studies three different local search mechanisms (c.f. localSearch() in Algorithm 5.4.1). They are all based on polynomial mutation.

- **Greedy Search**:

Algorithm 5.4.3 shows the first mechanism, Tabu Local Search (TLS). TLS creates Q mutants of a given strategy (d_i) using polynomial mutation and identifies the best of $Q+1$ strategies (d_i and its Q mutants). As mutants are created, TLS updates a tabu list T to record which parameters have been mutated so that a new mutant is never be created by altering taboo parameters (i.e., the parameters in T).

- **Tabu Search**:

  The second local search mechanism is called Greedy Local Search (GLS) (Algorithm 5.4.2). Similar to TLS, GLS creates Q mutants iteratively; however, it replaces the original strategy (e.g., d_i) with a mutant if the mutant wins over the original in a game (Line 7). Though Q iterations, GLS keeps the best mutant discovered so far and mutates it when mutation occurs.

- **Greedy Tabu Search**:

  Algorithm 5.4.4 shows the third local search mechanism, Greedy Tabu Local Search (GTLS). It customizes GLS with a tabu list T. It avoids taboo parameters in T when it performs mutation.

5.5 Experiment

This section evaluates BitC through simulations and discusses how BitC allows BSNs to adapt their configurations to given operational conditions (e.g., data request patterns placed by cloud applications and memory space availability in sink and sensor nodes).

5.5.1 Experiment Setting

Simulations are configured with the parameters shown in Table 26.

It assumes a nursing home where senior residents/patients live. A small-scale and a larger-scale simulations are carried out with 20 and 100 residents, respectively. The small-scale setting is used

unless otherwise noted. Each resident is simulated to wear four sensors: a blood pressure sensor, an ECG sensor and two accelerometers (Fig. 35).

Cloud applications issue 1,000 data requests during three hours. Data requests are uniformly distributed over virtual sensors. A time window is randomly set for each request to a sensor. For example, it is set with the uniform distribution in between 0 and 600 seconds for an ECG sensor (Table 26). Mutation rate is set to $1/V$ where V is the number of parameters in a strategy. Every simulation result is the average with 10 independent simulation runs.

Experiment assumes four types of residents (Table 27). 25% of residents are simulated to be in each category. Each resident wears two energy harvesters: piezoelectric energy generator (PEG) and thermoelectric generators (TEG). A PEG and a TEG are assumed to be embedded in a shoe and attached to the skin, respectively (Fig. 35). A PEG generates energy (piezoelectricity) from walking activities of a resident [PS05]. A TEG generates energy (thermoelectricity) from a resident's body temperature [Sal10].

The amount of harvested energy is computed based on a set of daily activities assumed for each type of residents. For example, a very healthy resident is assumed to have a scheduled walk and an excercise session with, for example, a treadmill under the average walking step frequency of 2 hertz. A PEG and a TEG is assumed to generate 11 mW per step and 0.06 mW per second [PS05, Sal10].

It simulates eight different combinations of constraints (Table 28): no constraints (C_∞), very lightweight (C_{VL}), lightweight (C_L), moderate (C_M), stringent (C_S), very stringent (C_{VS}), very stringent for energy consumption (C_{EN}) and very stringent for data yield (C_{DY}). C_∞ is used unless otherwise noted.

Comparative performance study is carried out for BitC's five variants (i.e., PD, HV, PD-HV, MM and PD-MM in Section 5.4.1). BitC-PD is used unless otherwise noted. BitC is also compared with NSGA-II, which is a well-known multiobjective evolutionary algorithm [DPA02]. BitC and NSGA-II use the same parameter settings shown in Table 26. All other NSGA-II settings are borrowed from [DPA02]. Both BitC and NSGA-II are implemented with jMetal [DNA10].

5.5.2 Experiment Results

Table 29 examines how a mutation-related parameter, called distribution index (η_m in [DPA02]), impacts the performance of BitC. This parameter controls how likely a mutated strategy is similar to its original. (A higher distribution index makes a mutant more similar to its original.) In Table 29, the performance of BitC is evaluated with the hypervolume measure that a set of dominant strategies yield in the 300th generation. The hypervolume metric indicates the union of the volumes that a given set of solutions dominates in the objective space [ZT98]. A higher hypervolume means that a set of solutions is more optimal. As shown in Table 29, BitC yields the best performance with the distribution index value of 60. (Local search is not used to obtain this result.) Thus, this parameter setting is used in all successive simulations.

Fig. 37 studies how different local search mechanisms impact the performance of BitC. It verifies that all of them can improve BitC's performance. Among them, greedy local search (GLS) is the most effective in both convergence speed and hypervolume. Thus, GLS is used in all successive simulations.

Table 30 illustrates the hypervolume that each BitC variant yields at the last generation. As shown in this table, BitC-HV yields the highest hypervolume value among five variants. Therefore, the variant is used in all successive simulations.

BitC yields a single set of objective values with dominant strategies at each generation while NSGA-II yields 100 sets of objective values with 100 individuals at each generation. Therefore, in Table 31, the BitC solution is evaluated against an NSGA-II individual that is closest to the solution in the objective space. Table 31 compares BitC-HV and NSGA-II based on three metrics: objective values, hypervolume and Euclidean distance. For NSGA-II, objective values are measured with an individual that minimizes the Euclidean distance to the BitC solution at the last generation. Hypervolume is measured with the NSGA-II individual. Distance is measured in between the NSGA-II individual and the BitC solution. Distance is computed with each objective normalized to [0, 1]. (The value range of distance is [0, 2].)

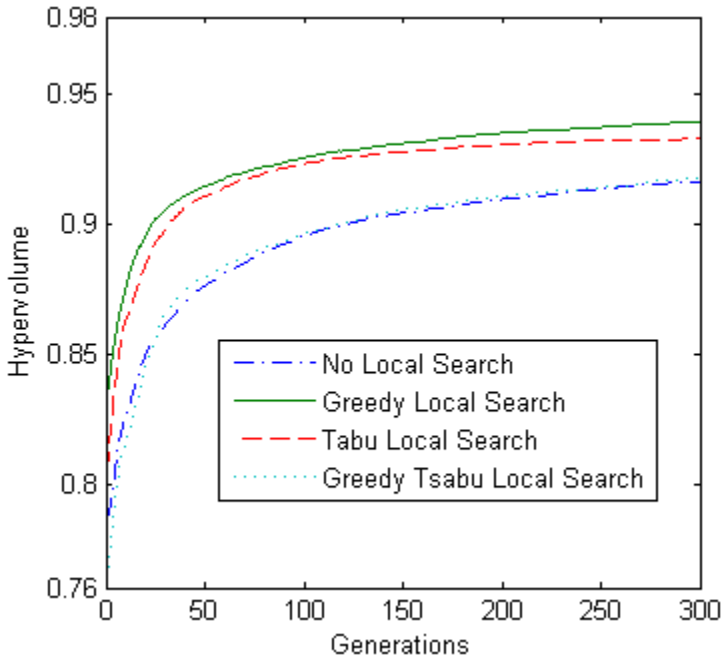


Figure 37: Local Search Comparison

As shown in Table 31, BitC-HV is non-dominated with NSGA-II with respect to four objectives. BitC-HV outperforms NSGA-II and it gets very close to the performance bounds in three objectives (request fulfillment, bandwidth consumption and energy consumption) while NSGA-II outperforms BitC-HV in data yield. BitC-HV yields a slightly higher (2% higher) hypervolume value than NSGA-II. Table 31 demonstrates that BitC-HV slightly outperforms NSGA-II on an a solution-to-solution basis. The performance bounds are given by running NSGA-II with single objective. Euclidean and Manhattan distances are used as metrics. In both metrics, a shorter distance means a given solution is closer to the performance bounds. BitC is closer to the bounds than NSGA-II by 29% and 34% in Euclidean and Manhattan distances, respectively.

Table 32 shows the variance of objective values that BitC-HV and NSGA-II yield at the last generation in 10 different simulation runs. A lower variance means higher stability (or higher similarity) in objective value results (i.e., lower oscillations in objective value results) among different simulation runs. BitC-HV maintains significantly higher stability than NSGA-II in all objectives

except energy consumption. On average, BitC's stability is 36.75% higher than NSGA-II's. This result exhibits BitC's stability property (i.e. reachability to at least one equillibira), which NSGA-II does not have.

Fig. 38 shows two three-dimensional objective spaces that plot a set of dominant strategies obtained from individual populations at each generation. Each blue dot indicates the average objective values that dominant strategies yield at a particular generation in 10 simulation runs. The trajectory of blue dots illustrates a path through which strategies evolve and improve objective values. Gray and red dots represent 10 different sets of objective values at the first and last generation in 10 simulation runs, respectively. While initial (gray) dots disperse (because the initial strategies are generated at random), final (red) dots are overlapped in a particular region. Consistent with Table 32, Fig. 38 verifies BitC's stability: reachability to at least one equilibria regardless of the initial conditions.

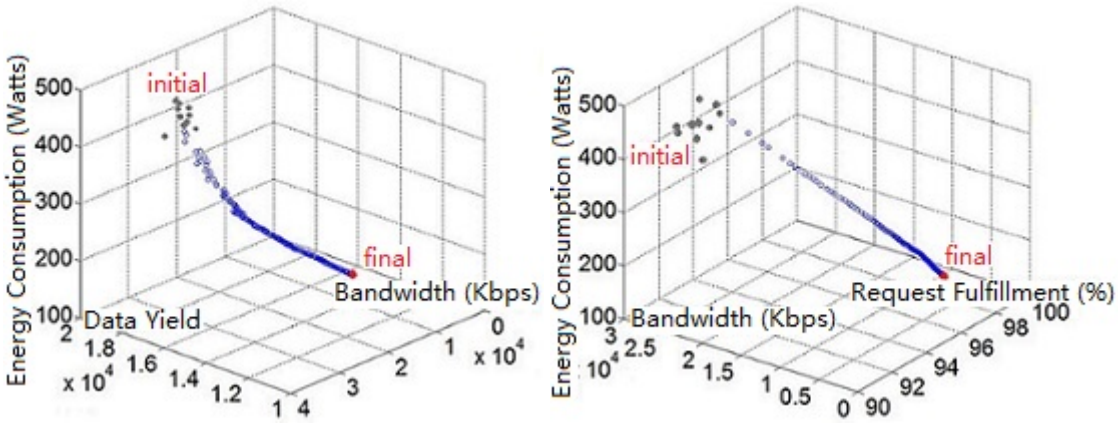


Figure 38: Three-dimensional Objective Spaces

Table 33 evaluates how different constraint combinations impact on the performance of BitC in objective values. The table shows the average, maximum and minimum objective values at the last generation subject to eight constraint combinations listed in Table 28. BitC successfully satisfies four constraint combinations (C_{VL},C_L, C_M and C_S). Under C_{EN} and C_{DY}, BitC fails to satisfy a constraint in each case although it satisfies three other constraints. And BitC fails to satisfy all

the objectives subject to C_{VS} constraints, since the constraints setting is very stringent. The best performance is produced under C_S. In most of the cases BitC fails to satisfy the data yield constraint due to it is a conflicting objective with other three, and BitC tries to balance the trade off among all the four objectives. Comparing the results of C_S and C_∞, and the results of C_S and C_{VS}, Table 33 demonstrates that BitC satisfies the data yield constraint by trading three other objectives for a global better performance. Given this result, C_S is used in all successive simulations.

Fig. 39 shows how BitC improves its performance through generations with 20 BSNs (i.e., 20 patients) and 100 BSNs. It shows the changes of objective values over generations. All four constraints are satisfied at the last generation. The two figures illustrate that BitC improves its objective values subject to given constraints by balancing the trade-offs among conflicting objectives. For example, BitC improves both request fulfillment and bandwidth consumption through generations while the two objectives conflict with each other.

Results are qualitatively similar comparing both results with 20 BSNs and 100 BSNs, although BitC yields a slightly lower performance with a larger number of BSNs. It is harder to satisfy given constraints in a larger-scale setting. (All four constraints are satisfied at the last generation.) BitC 100 BSNs's performance decreases a little respect to 20 BSNs in bandwidth consumption and energy consumption. It happens due to the increasing number of BSNs, BitC needs more bandwidth and energy to satisfy all BSNs needs. And as consequence the data yield increases, but 20 BSNs reaches slightly higher request fulfillment because it is easier for BitC to satisfy greater number of requests for smaller number 20 BSNs rather than 100 BSNs. Fig. 39 demonstrates that BitC scales well against the number of BSNs.

At the last generation BitC reaches hypervolume value 0.936 and 0.922 for 20 BSNs and 100 BSNs respectively.

Table 34 examines how BitC and NSGA-II maintain the lifetime of BSNs and yield data yield performance with energy harvesting enabled and disabled. Both BitC and NSGA-II utilize harvested energy to extend the lifetime of BSNs and in turn improve data yield performance. With energy harvesting enabled under a stringent set of constraints (C_S), BitC extends the BSN lifetime by 4.6%.

BitC maintains 39% longer lifetime than NSGA-II. Table 34 demonstrates that BitC successfully leverages energy harvesting to improve its performance.

5.6 Conclusion

This Chapter considers a layered push-pull hybrid communication for cloud-integrated BSNs and formulates a BSN configuration problem to seek optimal and stable solutions. An evolutionary game theoretic algorithm is used to approach the problem. Simulation results show that BitC seeks equilibria to perform adaptive and evolutionarily stable configuration strategies and adapt their configuration parameters to given operational conditions subject to given constraints. This study evaluates five algorithmic variants of BitC under various settings and demonstrates that BitC allows BSNs to successfully leverage harvested energy to balance their performance with respect to multiple objectives such as resource consumption and data yield. BitC's performance is evaluated in comparison to a well-known multiobjective evolutionary optimization algorithm, NSGA-II [DPA02], while maintaining 37% higher stability (lower oscillations) in performance across different simulation runs.

Algorithm 5.4.1: Evolutionary Process in BitC

1: $g = 0$
2: Randomly generate the initial N populations for N BSNs: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$
3: **while** $g < G_{max}$ **do**
4: **for** each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
5: $\mathscr{P}_i' \leftarrow \emptyset$
6: **for** $j = 1$ to $|\mathscr{P}_i|/2$ **do**
7: $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
8: $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
9: $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
10: $replica \leftarrow$ replicate($winner$)
11: **for** each parameter v in $replica$ **do**
12: **if** random() $\leq P_m$ **then**
13: $replica \leftarrow$ mutate($replica$, v)
14: **end if**
15: **end for**
16: $winner' \leftarrow$ performGame($loser$, $replica$)
17: $\mathscr{P}_i \setminus \{s_1, s_2\}$
18: $\mathscr{P}_i' \cup \{winner, winner'\}$
19: **end for**
20: $\mathscr{P}_i \leftarrow \mathscr{P}_i'$
21: $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
22: **while** d_i is infeasible **do**
23: $\mathscr{P}_i \setminus \{d_i\}$
24: $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
25: **end while**
26: $d_i \leftarrow$ localSearch(d_i)
27: Configure a BSN in question based on d_i.
28: **end for**
29: $g = g + 1$
30: **end while**

Algorithm 5.4.2: Greedy Local Search (localSearch())

Require: d_i: Dominant strategy to improve

Ensure: Improved dominant strategy

1: **for** $i = 1$ to Q **do**
2: **for** each parameter v in $replica$ **do**
3: **if** random() $\leq P_m$ **then**
4: $replica \leftarrow$ mutate(d_i, v)
5: **end if**
6: **end for**
7: $d_i \leftarrow$ performGame($replica$, d_i)
8: **end for**
9: **return** d_i

Algorithm 5.4.3: Tabu Local Search (localSearch())

Require: d_i: Dominant strategy to improve

Ensure: Improved dominant strategy

1: $T \leftarrow \emptyset$
2: **for** each parameter $v \in d_i$ and $v \notin T$ **do**
3: **if** random() $\leq P_m$ **then**
4: $replica \leftarrow$ mutate(d_i, v)
5: $T \leftarrow T \cup \{v\}$
6: **end if**
7: **end for**
8: **for** $i = 1$ to $Q - 1$ **do**
9: **for** each parameter $v \in d_i$ and $v \notin T$ **do**
10: **if** random() $\leq P_m$ **then**
11: $replica' \leftarrow$ mutate(d_i, v)
12: $T \leftarrow T \cup \{v\}$
13: **end if**
14: **end for**
15: $replica \leftarrow$ performGame($replica$, $replica'$)
16: **end for**
17: $best \leftarrow$ performGame($replica$, d_i)
18: **return** *best*

Algorithm 5.4.4: Greedy Tabu Local Search (localSearch())

Require: d_i: Dominant strategy to improve

Ensure: Improved dominant strategy

1: $T \leftarrow \emptyset$
2: **for** $i = 1$ to Q **do**
3: **for** each parameter $v \in d_i$ and $v \notin T$ **do**
4: **if** random() $\leq P_m$ **then**
5: $replica \leftarrow$ mutate(d_i, v)
6: $T \leftarrow T \cup \{v\}$
7: **end if**
8: **end for**
9: $d_i \leftarrow$ performGame($replica$, d_i)
10: **end for**
11: **return** d_i

Table 26: Body Sensor Networks Simulation Settings

Parameter	**Value**
Duration of a simulation (W)	10,800 secs (3 hrs)
Number of simulation runs	10
Number of BSNs (N)	20 and 100
Number of sensor nodes in a BSN (M)	4
Memory space in a sensor node	2 GB
Memory space in a sink node	16 GB
Total number of data requests from cloud apps	1,000
Size of a data request (d_r and d'_r)	100 bytes
Size of an error message (e_r)	250 bytes
Energy consumption for a single bit of data (e_t)	0.001 Watt
Blood pressure request time window	[0, 1000 secs]
Acelerometer request time window	[0, 1800 secs]
ECG request time window	[0, 600 secs]
Number of generations (G_{max})	300
Number of local search iterations (Q)	20
Population size ($\|\mathscr{P}_i\|$)	100
Mutation rate (P_m)	$1/V$

Table 27: Energy Harvesting Configurations

Category	Energy source	Harvested energy in 3 hrs	Total harvested energy in 3 hrs
Very healthy	Piezo (2.0 Hz) Thermo	18.27 W 0.10 W	18.37 W
Healthy	Piezo (1.38 Hz) Thermo	12.56 W 0.068 W	12.63 W
Rehabilitation	Piezo (0.25 Hz) Thermo	2.28 W 0.0124 W	2.29 W
Wheelchair	Piezo (0.0 Hz) Thermo	0.0 W 0.0 W	0.0 W

Table 28: Constraint Combinations

Constraint Combination	C_E (W)	C_Y	C_R $(\%)$	C_B (Kbps)
C_∞	∞	0	0	∞
C_{VL}	450	16	90	30
C_L	350	17	93	25
C_M	200	18	95	20
C_S	150	19	97	10
C_{VS}	100	20	99	7
C_{EN}	50	16	90	30
C_{DY}	450	25	90	30

Table 29: Impacts of Distribution Index Values on Hypervolume (HV)

Distribution Index	**HV**	**Distribution Index**	**HV**
45	0.886	50	0.910
55	0.912	60	0.917
65	0.903		

Table 30: Comparison of BitC's Variants in Hypervolume

Algorithm	Hypervolume (HV)
BitC PD	0.9247
BitC HV	0.9394
BitC PD-HV	0.9056
BitC MM	0.9071
BitC PD-MM	0.9143

Table 31: Comparison of BitC-HV and NSGA-II

Objective	NSGA-II	BitC-HV	Bounds
Request Fulfillment: f_R (%)	97.6	98.35	99.00
Bandwidth: f_B (Kbps)	10.45	7.41	7.32
Energy consumption: f_E (Watts)	178.89	129.26	126.91
Data Yield: f_Y	37.92	14.54	44.72
Hypervolume	0.922	0.9394	-
Euclidean distance	0.152	0.108	-
Manhattan distance	0.232	0.153	-

Table 32: Stability of Objective Values in BitC-HV and NSGA-II

Objectives	BitC-HV	NSGA-II	Diff (%)
Request Fulfillment: f_R	0.05	0.5	90%
Bandwidth: f_B	0.17	0.22	22.72%
Energy Consumption: f_E	3.38	3.01	-12.33%
Data Yield: f_Y	0.42	0.97	56.70%
Average Difference (%)	–	–	36.75%

Table 33: Objective Values of BitC-HV under Different Constraint Combinations

Constraint Combination		f_B (Kbps)	f_E (W)	f_R(%)	f_Y
C_∞	maximum	11.38	195.81	97.7	30.25
	Avg	11.07	190.28	97.57	27.33
	minimum	10.88	186.96	97.4	25.54
C_{VL}	maximum	12.11	208.96	97.3	17.64
	Avg	11.8	203.77	97.27	16.62
	minimum	11.53	198.94	96.9	15.64
C_L	maximum	12.45	214.72	97.1	14.48
	Avg	12.11	209.72	97.04	14.32
	minimum	11.87	205.84	97	14.19
C_M	maximum	9.13	159.93	97.85	17.53
	Avg	9.04	157.98	97.81	16.39
	minimum	8.90	155.14	97.7	15.61

Constraint Combination		f_B (Kbps)	f_E (W)	$f_R(\%)$	f_Y
C_S	maximum	7.69	133.37	98.4	14.86
	Avg	7.41	129.26	98.35	14.54
	minimum	7.25	126.68	98.3	14.02
C_{VS}	maximum	9.34	161.09	97.9	15.45
	Avg	9.24	159.48	97.88	14.95
	minimum	9.05	156.81	97.85	14.59
C_{EN}	maximum	12.08	208.63	96.95	16.50
	Avg	11.81	204.66	96.94	15.87
	minimum	11.58	201.50	96.9	15.00
C_{DY}	maximum	11.89	206.14	97.45	15.41
	Avg	11.32	195.72	97.32	14.75
	minimum	10.93	188.75	97.2	13.91

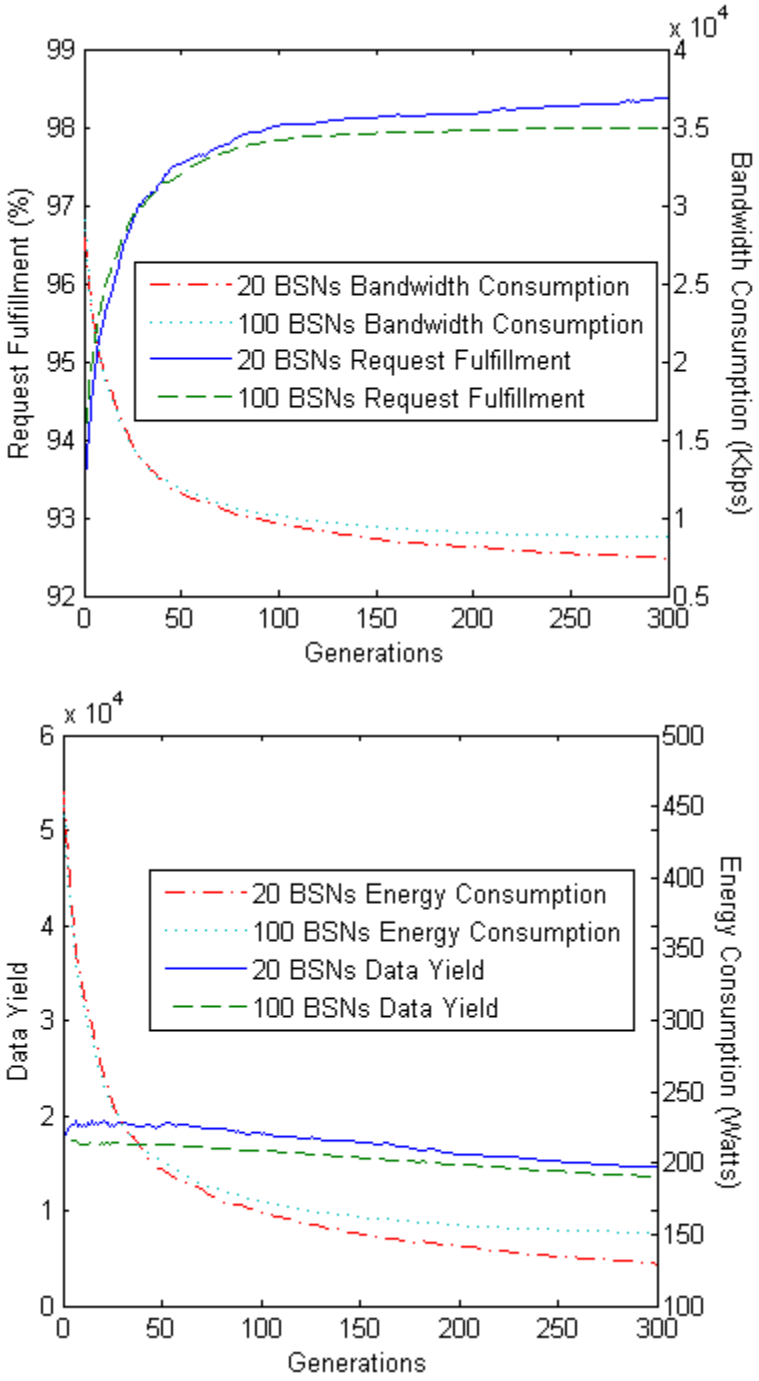


Figure 39: 20 BSNs and 100 BSNs performance comparison

Table 34: Comparison of BitC-HV and NSGA-II in BSN Lifetime and Data Yield with Energy Harvesting (EH) Enabled and Disabled

Algorithms		Lifetime (hrs)	Data yield (f_Y)
C_∞	BitC-HV w/o EH	2.36	21,505
	BitC-HV w/ EH	2.47 (+4.66%)	22,507
	NSGA-II w/o EH	0.66	4,702
	NSGA-II w/ EH	0.69 (+4.55%)	4,916
C_S	BitC-HV w/o EH	3.48	16,866
	BitC-HV w/ EH	3.64 (+4.60%)	17,641
	NSGA-II w/o EH	2.51	31,727
	NSGA-II w/ EH	2.62 (+4.38%)	33,117

Table 35: Notation table

Notation	Description	Notation	Description
B	set of BSNs	P_i	population size
N	number of BSNs	Q	number of local search iterations
M	number of sensors	f_R	request fulfillment objective
m	sink node	f_B	bandwidth objective
O	data transmission interval	f_E	energy consumption objective
p	sensing interval	f_Y	data yield objective
q	sampling rate	C_R	request fulfillment constraint
R	data requests	C_B	bandwidth constraint
K	set of strategies	C_E	energy consumption constraint
G	number of generations	C_Y	data yield constraint
P_m	mutation rate		

CHAPTER 6

MOLECULAR COMMUNICATION

Molecular communication is a new research area where researches focus on the design of a communication system where transmission and reception of information are carried by molecules. It is used in many places such as intra-body sensor networks, lab-on-a-chip, drug delivery systems, so on. This Chapter describes a multi-objective multi-hub molecular communication optimization problem where two objectives Round Trip Time (RTT) and message delivery success rate are considered to be optimized. It proposes and evaluates a EGTMOA framework based algorithm called Evolutionary Multi-hub Molecular Communication Optimizer (EMMCO) that aims to solve the formulated multi-objective optimizition problem. Experiments are performed in a wide range of simulation settings. And results show that EMMCO successfully balances the trade-off of RTT and success rate while optimizing their performance.

6.1 Introduction

If we look back to the computer history, one of the main trend of computing system evolution is the size reduction of electronic devices and computing units. From the first computer ENIAC (1946) to personal computer (1975), few years later the appearance of the first laptop (1981), then smart phones become popular in late 90s, and in the last 20 years MEMS (Micro Electro Mechanical Systems) devices that are made up of components between 1 and 100 mm in size. So, what is now ? Recent researches are start focusing on reducing computing and transmission units even smaller,

down to nanoscale sizes between $nm - \mu m$. Those nanoscale units are called nanomachines. In this research we target Biological Nanomachines in particular.

Bio-nanomachines are the most basic functional units in biological nanoscale systems. They are made from modified natural cells with artificial functions. Examples of some artificial functions:

- Synthesizing and releasing specific molecules to the environment.
- Capturing molecules from the environment.
- Logic gates to trigger programmed chemical responses upon receiving specific molecules.
- Toggle switches (1-bit memory).
- Queuing of molecules with nested vesicles.
- Oscillators (clocks).

When nanomachines are networked they could perform collaborative tasks that no individual nanomachines could. In order to network nanomachines we need a nanoscale communication system which is call molecular communication. Molecular communication systems use molecule to digitally encode and decode messages. Later those information carrier molecules are delivered into communication media for transmission. The communication media could be anything where molecules could propagate, air, water, human body, so on. Molecular communication presents many advantages over traditional communication system such as RF, Satellite.

1. It is not subject to requirement of using antennas that are sized to a specific ratio of the signal wavelength.
2. It requires very little energy consumption.
3. It can be made bio-compatible, and transmission can occurs inside human body.

Molecular communication is used in many different fields i.e.:

- **Lab-on-a-chip**: Chemical analysis of biological samples for diagnosis of diseases biometric authentication and other purposes. It Provides functionalities to manipulate molecules on a single chip such as transporting molecules to specific locations, Mixing one type of molecules with another type of molecules, and Separating specific types of molecules from a mixture of molecules.

- **Artificial morphogenesis**: Artificial morphogens (AMs) are molecules that encode artificial morphological information. It affects the growth and differentiation of AM recipients into specific spatial structures by controlling the patterns of AM propagation. Artificial morphogenesis target areas like tissue engineering and regenerative medicine.

- **Drug delivery**: Molecular communication helps to guide drug molecules into the site of diseased cells (e.g., tumor site) in order to minimize possible drug side effects.

This study investigates an EGTMOA framework based algorithm EMMCO that aims to improve the performance of latency (RTT) and reliability (success rate) in a multi-hub molecular communication system. It focus on intra-body molecular communication environment.

6.2 Problem Formulation

In this study we consider a short-range (up to 100 m) molecular communication where bio-nanomachines transmit and receive molecule-encoded messages and applies Stop-and-Wait Automatic Repeat Request (SW-ARQ) for feedback-based reliable communication in noisy intra-body environments. Three coommunication transports are considered.

- **Diffusive transports :** molecules propagate from transmitter (Tx) to the receiver (Rx) via random thermal motion Fig. 40.

  Where Information molecules are modified DNA molecules and noise molecules are stationary obstacles for information molecules to move. In diffusive communication no infrastruc-

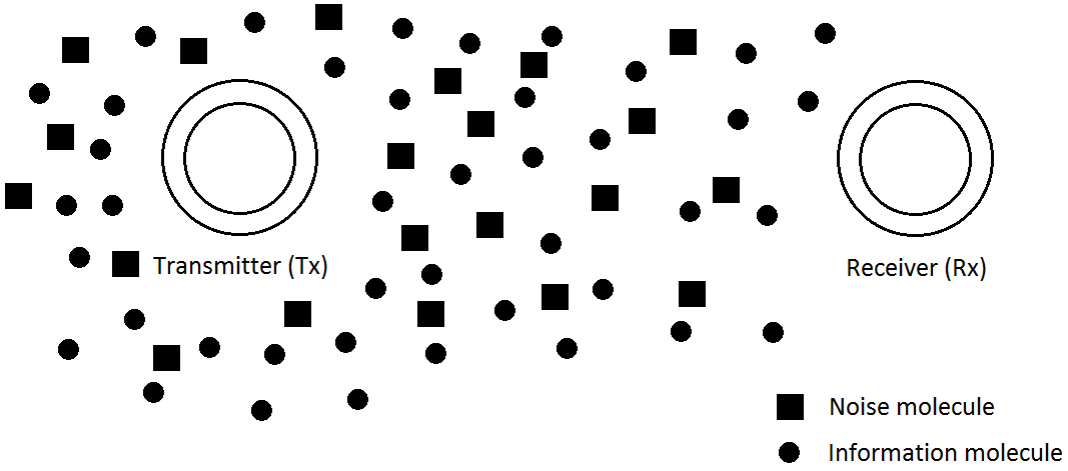


Figure 40: Diffusive communication

ture is needed and very little energy is required. The drawback is that it is very slow and it just work on short distance between Tx and Rx.

- **Directional transports :** Molecules directionally move from the transmitter to receiver on predefined microtubule by using molecular motors Fig. 41. Information molecules could diffuse away from a microtubule randomly and when colliding with noise molecules. While diffusing they may contact again a microtubule and re-walk along the microtubule.

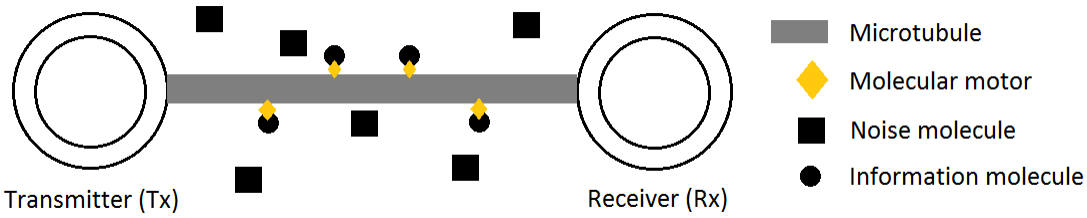


Figure 41: Directional communication

Directional communication is faster and can carry longer communication distance. But it requires infrastructure and more energy consumption.

- **Hybrid transports :** It is just a simple combination of above two mentioned communication type where molecules propagate with both diffusive and directional transports.

In a real molecular communication environments they are many factors that lead to the losses and out of sequence deliveries of information that molecules carry. Those factors could be stochastic molecular propagation, molecule collisions, noisy environments, so on. This study employ a Stop-and-Wait Automatic Repeat Request (SW-ARQ) mechanism to address this reliability problem.

In SW-ARQ an error-control method to ensure that all messages are delivered from Tx to Rx in the correct order. A simplest ARQ method is feasible enough to realize with simple computation and memory functions in modified biological cells. The concept of SW-ARQ in molecular communication is the same as in a TCP connection where an acknowledgment data is used to indicate that the receiver has successfully received the information data and the transmitter should starts sending the next message. Fig 42 illustrates a two channel molecular communication. One for transmitter to propagate information molecules to receiver, and another for receiver to propagate acknowledgment (ACK) molecules to transmitter.

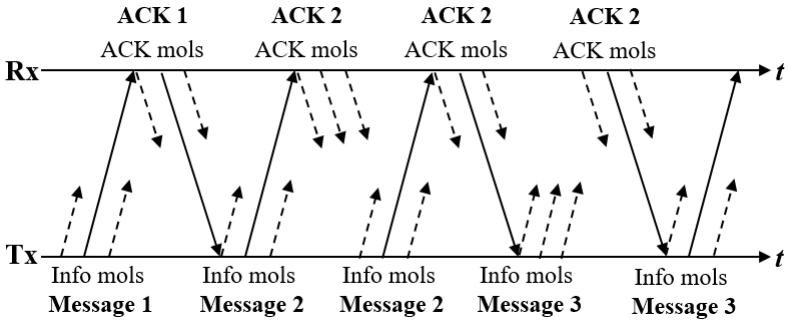


Figure 42: Stop-and-Wait Automatic Repeat Request communication protocol

In Fig 42 there are multiple (n) information molecules per message that are sent from transmitter to receiver. Receiver upon receives the information molecule successfully, it returns back an ACK molecule to the sender. And if the sender successfully receives the ACK molecule from the previous message then it starts to send the next set of information molecule for the next message, and so on.

6.2.1 Multi-Hub Molecular Communication Simulator

Human body is a complex communication media, any chemical body composition could be a noisy factor in the molecular communication. Thus, longer distance higher chance the message molecule collide and to be destroyed by an obstacle, and as consequence less change for the receiver to receive successfully the information sent by transmitter. To ensure the success rate we employ a multi-hub architecture Fig. 43. Intermediate nodes are places in between the transmitter and receiver. Their job are to repeat the transmission sent by the transmitter node once a message molecule is received and to propagate ACK molecules for such received message molecule. They act as an inter-median station in a molecular communication to increase a long distance communication success rate.

In this study the whole molecular communication is done through a multi-hub molecular communication simulator that is implemented as an extension based on the previous work [Jon]. In [Jon], the simulator is designed to handle a simple molecular communication with just one transmitter and one receiver. However in this study we consider a multi-hub molecular communication architecture as illustrated in Fig. 43.

Multi-Hub Molecular Communication Simulator simulates an intra-body molecular communication environment. The major components are:

- **Transmitter node** transmits n info molecules at a time for a single message delivery and it waits for at least one of n ACK molecules during the retransmission timeout interval (RTO). During RTO, transmitter never sends out the next set of info molecules. It transmits the next set of n info molecules once receiving at least one of ACK molecules from receiver or an intermediate node. It retransmits the n info molecules if the transmitter does not receive at least one of n ACK molecules in RTO or receives a retransmitted ACK(s).

- **Receiver node** transmits n ACK molecules in response to receiving at least one of n info molecules and it waits for the next set of info molecules from transmitter during RTO. It retransmits the n ACK molecules if receiver does not receive at least one of n info molecules in RTO or receiver receives a retransmitted info molecule(s).

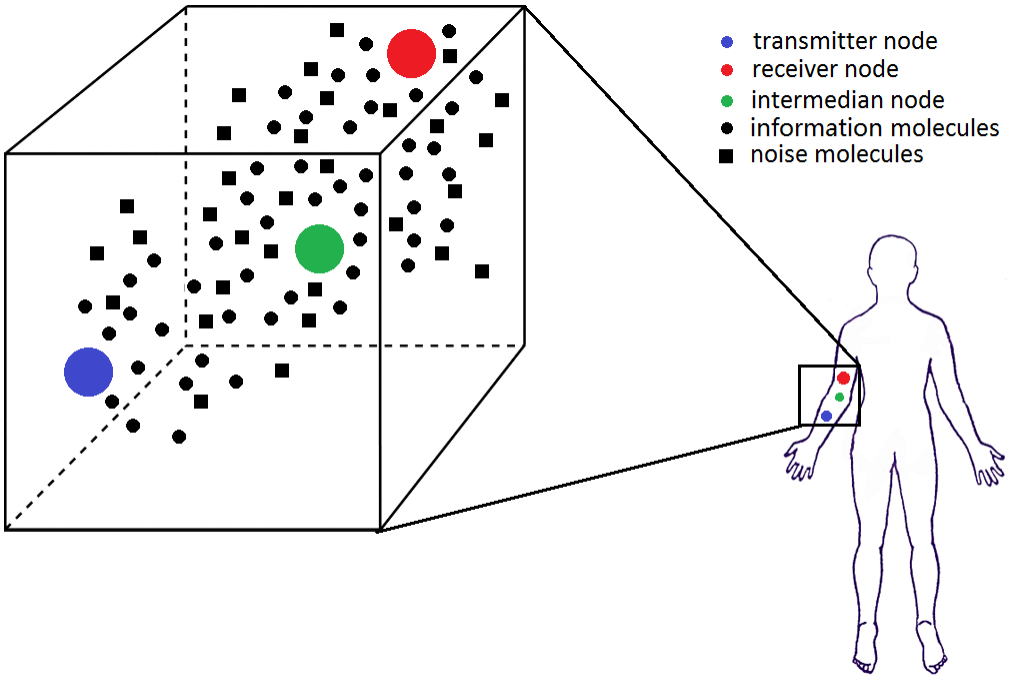


Figure 43: Illustration of a multi-hub intra-body molecular communication

- **Intermediate node** acts as an inter-median station in between transmitter and receiver. It just behaves as a combination of a transmitter node and a receiver node.
- **Information molecules** are message carriers with mission to be delivered in the receiver node.
- **Noise molecules** are random obstacles that exist in an intra-body environment. They could be any human body chemical composition or residues from ACK or INFO molecules of previous already delivered messages that now are randomly wandering in the environment.

We consider a communication failure if any node could not get a response during RTO after m number of retransmission is done. The simulator simulates the random transmission process of a multi-hub molecular communication where each molecules moves in a random direction in each simulation defined time unit. In this study we aim to optimize two objectives.

- Round Trip Time (RTT): It is the average time steps from transmitter sending out information molecules to successfully receiving ACK molecules.
- Transmission success rate: It is the percentage of delivered message respect to the total number of transmitted messages.

The goal is to minimize RTT while maximizing success rate. They both are computed through the multi-hub molecular communication simulator.

6.3 EMMCO

This section describes an EGTMOA framework based algorithm called Evolutionary Multi-hub Molecular Communication Optimizer (EMMCO) that aims to optimize RTT and success rate in a multi-hub molecular communication environment. EMMCO divides the entire optimization problem into N subproblems, where $N = M + 2$ and M is the number of intermediate nodes, plus two because of transmitter and receiver. EMMCO maintains N populations, $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$, for $M + 2$ nodes and performs games among strategies in each population. Each strategy $s(i)$ specifies a particular configuration for a node i using four types of parameters: number of information molecules and acknowledgment per message and number of retransmissions (n_i and m_i) as well as retransmission time out (RTO) and the communication protocol used (diffusive, directional or hybrid) (r_i and p_i).

$$s(i) = (n_i, m_i, r_i, p_i) \;\; 1 < i < N \tag{6.1}$$

Each objective evaluation is computed through the multi-hub molecular communication simulator described in Section 6.2.1, which is a very expensive process. We try to minimize the time of objective evaluation by building an algorithm that uses a very basic EGTMOA framework. Unlike Cielo and BitC, EMMCO does not perform elitism or local search and it uses a simple Hypervolume

calculation as quality indicator. EMMCO attempts to seek toward an optimal solution that minimize RTT and increase success rate while ensuring a tolerable computing speed.

Algorithm 6.3.1 shows how EMMCO seeks an evolutionarily optimal configuration strategy for each node through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$ (Line 2). In each generation (g), a series of games are carried out on every population (Lines 4 to 22). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to performance objectives described in Section 6.2 (Lines 7 to 9). The winner is replicated to increase its population share and mutated with polynomial mutation (Lines 10 to 18) [DPA02]. Mutation randomly chooses a parameter (or parameters) in a given strategy with a certain mutation rate P_m and alters its/their value(s) at random (Lines 12 to 14). And the loser disappears in the population.

Once all strategies play games in the population, EMMCO identifies a strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 20). In the end, EMMCO configures a node with the parameters contained in the dominant strategy (Line 21).

A game is carried out based on the superior-inferior relationship between given two strategies (c.f. performGame() in Algorithm 5.4.1). In EMMCO we use Hypervolume (HV) as quality indicator. HV measures the volume that a given strategy s dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{6.2}$$

Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

6.4 Experiment

This section evaluates EMMCO through simulations and discusses how EMMCO allows different nodes in molecular communication to adapt their configurations in a random noisy intra-body environment.

6.4.1 Experiment Setting

Simulations are configured with the parameters shown in Table 36.

It assumes a noisy intra-body environment with dimension $150\mu m$ x $150\mu m$ x $150\mu m$, where S noises molecules are randomly placed in the environment. A multi-hub molecular communication system is considered to deliver 100 messages from a transmitter to a receiver in distance of D through M number of intermediate nodes. Intermediate nodes are placed in an equivalent distance between transmitter and receiver. Time steps are defined as seconds in the simulation, in each time step each molecule move to a random direction. An information molecule is destroyed if it collides against a noise molecule during the transmission.

The simulation computes RTT as the average of time steps (seconds) that takes from a message is sent out to receiving its ACK molecule back. If any of $M+2$ node do not receive a response (INFO or ACK molecule) after a number of retransmissions, then it considers a message is lost. Thus, we compute the success rate as the percentage of successfully delivered messages. Experiment are carried out with different distances and noise levels (number of noise molecules) combination.

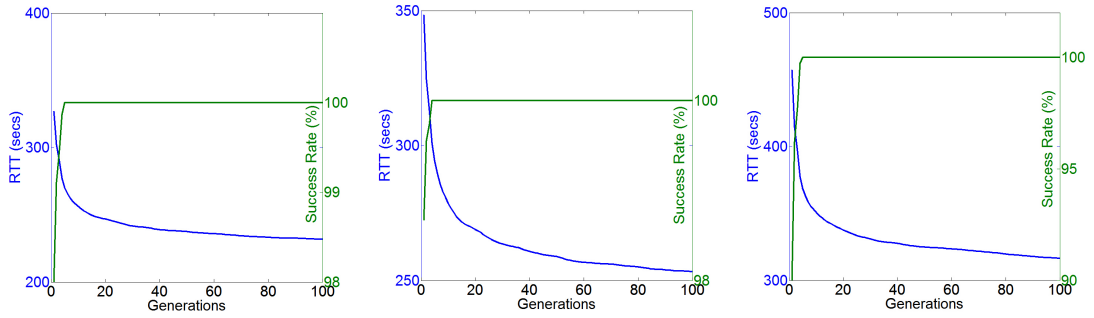
6.4.2 Experiment Results

Results are compared against a Random Search approach which is a mechanism that randomly select a configuration for each of the node and to perform the simulation under the same experiment setting as EMMCO. Fig. 44 to 46 show RTT and success rate performance improvement through 100 generations with three different distance between transmitter and receiver $30\mu m$, $50\mu m$, and $90\mu m$

respectively. Each of them is composed by three sub-figures that represent objective performances under three different noise level setting, 0, 100,000, and 338,000 noise molecules respectively. The number 338,000 is chosen because it occupies around 10% of the entire experiment space of dimension $150\mu m$ x $150\mu m$ x $150\mu m$. All objective values are computed as average of 10 simulation runs. A maximum tolerated simulation time of 2 days is set as the upper cap of running time per run. Any simulation run beyond this value is automatically stopped, so no results is obtained from runs that are longer than 2 days.

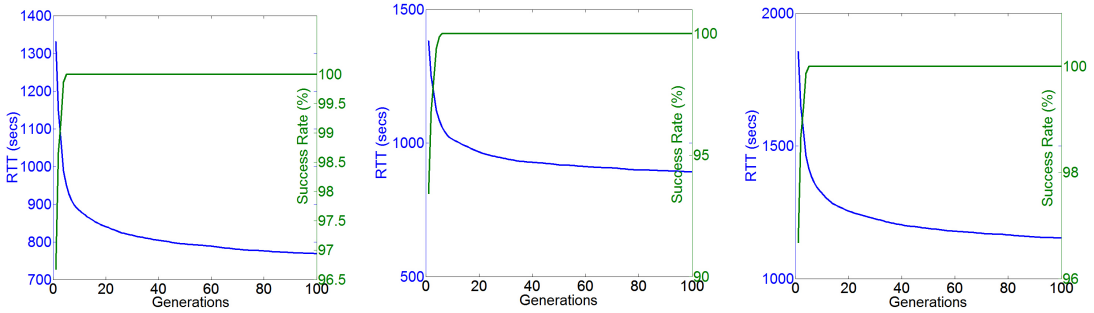
Fig. 44 to 46 illustrate that regardless of the distance EMMCO is able to reach 100% success rate very fast in the first 10 generations under three levels of noises, and RTT decreases efficiently through generations. It makes sense that longer the distance and higher noise level are, slower RTT is. From Fig. 44 to 46 we can conclude that RTT and success rate get a performance improvement (from generation 1 to 100) in between $39\% - 78\%$ and $2\% - 10\%$ respectively. No result is shown for a noise level 338K and distance $90\mu m$ because its running time is higher than the maximum tolerated running time 2 days. Results show that EMMCO successfully optimizes both RTT and success rate under different environment settings.

Performance comparison of EMMCO and Random Search are shown in the table 37 where R_{RTT} is the RTT performance value computed with Random Search and E_{RTT} is the RTT performance value got from the last generation using EMMCO. The same applies to success rate, R_{SR} is the success rate value computed using Random Search and E_{SR} is the success rate value got from the last generating using EMMCO. Runtime shows the average running time of EMMCO across different runs. They are all calculated as the average of 10 simulation runs. Some observations delivered from results are that longer distance and higher noise level require longer RTT. In all settings we could reach 100% success rate using EMMCO. No results are shown for distance $90\mu m$ with 338k noise molecules because its running time is higher than 2 days. Results show that EMMCO outperform Random Search in any experiment settings, RTT and success rate gain performance in between $27\% - 44.6\%$ and $3\% - 11\%$ respectively.



(a) 0 noise molecules (b) 100k noise molecules (c) 338k noise molecules

Figure 44: Objective Values of EMMCO with distance between transmitter and receiver $30\mu m$

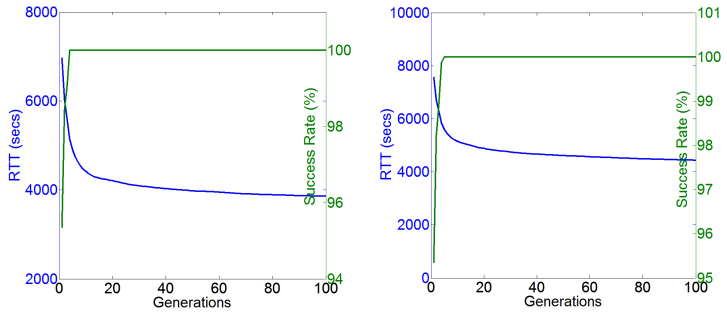


(a) 0 noise molecules (b) 100k noise molecules (c) 338k noise molecules

Figure 45: Objective Values of EMMCO with distance between transmitter and receiver $50\mu m$

6.5 Conclusion

This Chapter considers a noisy intra-body multi-hub molecular communication and formulates a molecular communication configuration problem to seek a optimal solution that improves RTT and success rate simultaneously. An evolutionary game theoretic algorithm EMMCO is used to approach the problem. Simulation results show that EMMCO allows molecular communication successfully configures its nodes to improve their performance with respect to multiple objectives round trip time and message delivery sucess rate. EMMCO reaches 100% success rate on message



(a) 0 noise molecules (b) 100k noise molecules (c) 338k noise molecules

Figure 46: Objective Values of EMMCO with distance between transmitter and receiver $90\mu m$

delivery regardless of the environment setting. EMMCO's performance is evaluated in comparison to a Random Search approach. Results conclude that EMMCO performs better in both RTT and success rate than Random Search with range of $(27\% - 44.6\%)$ and $(2.3\% - 11\%)$ respectively across different simulation runs.

Algorithm 6.3.1: Evolutionary Process in EMMCO

$g = 0$
Randomly generate the initial N populations for N BSNs: $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, ..., \mathscr{P}_N\}$
while $g < G_{max}$ **do**
  for each population $\mathscr{P}_i$ randomly selected from $\mathscr{P}$ **do**
    $\mathscr{P}_i' \leftarrow \emptyset$
    for $j = 1$ to $|\mathscr{P}_i|/2$ **do**
      $s_1 \leftarrow$ randomlySelect($\mathscr{P}_i$)
      $s_2 \leftarrow$ randomlySelect($\mathscr{P}_i$)
      $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
      $replica \leftarrow$ replicate($winner$)
      for each parameter v in $replica$ **do**
        if random() $\leq P_m$ **then**
          $replica \leftarrow$ mutate($replica$, v)
        end if
      end for
      $\mathscr{P}_i \setminus \{s_1, s_2\}$
      $\mathscr{P}_i' \cup \{winner, replica\}$
    end for
    $\mathscr{P}_i \leftarrow \mathscr{P}_i'$
    $d_i \leftarrow argmax_{s \in \mathscr{P}_i} x_s$
    Configure a node in question based on d_i.
  end for
  $g = g + 1$
end while

Table 36: Molecular Communication Simulation Settings

Parameter	**Value**
Environment dimension	$150\mu m$ x $150\mu m$ x $150\mu m$
Number of intermediate nodes (M)	5
Total number of nodes (N)	7
Number of noise molecules (S)	$(0, 100k, 338k)$
Diameter of Transmitter and Receiver	$5\mu m$
Diameter of an info/ACK molecule	$1\mu m$
Distances between transmitter and receiver (D)	$(30\mu m, 50\mu m, 90\mu m)$
Number of messages	100
Diffusion coefficient	0.5
Molecular movement speed	$1\mu m/s$ (constant)
Derailing probability	0.25
Number of simulation runs	10
Number of generations (G)	100
Population size ($\lvert\mathscr{P}_i\rvert$)	50
Mutation rate (P_m)	$1/V$
Maximum tolerated simulation time per run	2 days

Table 37: Performance comparison of EMMCO and Random Search

Distance	$30\mu m$			$50\mu m$			$90\mu m$		
Noise	0	100k	338k	0	100k	338k	0	100k	338k
$R_{RTT}(s)$	326	348	457	1331	1382	1855	6968	7579	-
$E_{RTT}(s)$	231	253	316	768	891	1153	3857	4436	-
$Gain(\%)$	29	27	31	42.3	35.5	37.8	44.6	41.4	-
$R_{SR}(\%)$	97	97	89	95	92	95	94	94	-
$E_{SR}(\%)$	100	100	100	100	100	100	100	100	-
$Gain(\%)$	3	3	11	5	8	5	6	6	-
Runtime(h)	0.7	3.3	10.7	2.5	8.8	26.7	10.3	31.3	> 48

CHAPTER 7

CONCLUSION

This dissertation proposes and evaluates an Evolutionary Game Theory framework based Evolutionary Algorithm called EGTMOAs which is a meta-heuristic approximation method that seeks the set of dominant strategies as a whole solution. A theoretical analysis proves that EGTMOAs guarantee to provide a high quality and stable solution in a reasonable running time.

Three different multi-objective applications are studied in this dissertation, Cloud Virtual Machine placement, Body sensor network configuration and Multi-hub Molecular communication configurations. Different variants of EGTMOA algorithms are proposed to solve each problem and many experiments are performed with a wide variety of simulation settings to evaluate EGTMOA's performance. Evaluations mainly target on three respects of EGTMOAs quality, stability and running time.

Simulation results verify the theoretical analysis by showing that EGTMOAs successfully balance the trade-off of conflicting objectives while improving their performance simultaneously and outperform many well known multi-objective and meta-heuristic algorithms in quality, stability, convergence speed, and running time.

CHAPTER 8

FUTURE DIRECTIONS

There are many potential future directions originated from this dissertation.

8.1 Noise Handling

Multi-hub molecular communication simulator describes a random intra-body transmission process. Objective values computed through a random process are not reliable, in other word they are noisy. Those values are different under the same simulation settings in different runs. In this thesis I use average to evaluate objective values across different runs, however it may not be accurate. A noise handling in optimization process is needed, I propose as a future work to employ the notion of Depth which is a statistical operator that could be integrated for robust dominance comparison. Depth considers finding the media value in a multi-objective spaces.

8.2 Speeding Up

Objective evaluations are quite expensive in some cases, i.e. molecular communication. I propose as a future direction to integrate EGTMOAs with Neural networks for approximating objective values instead of using real objective functions. The basic idea is to use objective functions in the first 100 generations and to record the pair of decision variable vectors and objective values as features and label respectively in a dataset. Then we train a neural network to have a good approximation model from the dataset, after 100 generations we could use the model to compute objectives rather than using expensive objective functions. Of course we should consider the approximation error using

neural network, and training time. It may not be feasible if we could not find a good approximation model or the training time is longer than the original EGTMOA running time.

8.3 Fairness

EGTMOA seeks a set of dominant strategies as a global solution and it treats all players in the same manner. But sometimes players may have different requirements, for instance in the body sensor network problem a heart diseases patient may need more attention than a patient who is doing rehabilitation from appendectomy. Therefore higher amount of resources is needed for a heart diseases patient. In order to overcome this fairness problem, I propose to use the notion of coalition. Same type of players are grouped into a coalition, and each coalition has its own objective functions to be optimized. The main idea is to perform cooperate games among players of the same coalition, they attempt to maximize their payoff as a global objective. And to perform competitive games among coalitions, each coalition tries to maximize its own payoff based on its requirement (constraints).

8.4 Cloud simulator extension

Several extensions are planned as future work in the cloud virtual machine placement problem. One of them is to allow each application to consist of an arbitrary number of servers and operate on an arbitrary number of VMs. (Currently, each application is assumed to consist of three servers and operate on three VMs.) Another extension is to run simulations with publicly available simulators such as CloudSim [CRB11].

REFERENCE LIST

[ACC10] A. Ambrose, M. Cardei, and I. Cardei. "Patient-centric hurricane evacuation management system." In *29th IEEE Int'l Performance Computing and Communications Conference*, 2010.

[AG11] Khandakar Entenam Unayes Ahmed and Mark A. Gregory. "Integrating Wireless Sensor Networks with Cloud Computing." In *7th International Conference on Mobile Ad-hoc and Sensor Networks*, 2011.

[AHS07] K. Aberer, M. Hauswirth, and A. Salehi. "Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks." In *Proc. the 8th IEEE Int'l Conference on Mobile Data Management*, 2007.

[BBS12] M. Boulmalf, A. Belgana, T. Sadiki, S. Hussein, T. Aouam, and H. Harroud. "A lightweight middleware for an e-health WSN based system using Android technology." In *Int'l Conference on Multimedia Computing and Systems*, 2012.

[BDF03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. "Xen and the art of virtualization." In *Proc. of ACM symposium on operating systems principles*, October 2003.

[BHS07] P. Boonma, Q. Han, and J. Suzuki. "Leveraging Biologically-inspired Mobile Agents Supporting Composite Needs of Reliability and Timeliness in Sensor Applications." In *Proc. IEEE Int'l Conf. on Frontiers in the Convergence of Biosci. and Info. Tech.*, 2007.

[BS10] P. Boonma and J. Suzuki. "TinyDDS: An Interoperable and Configurable Publish/Subscribe Middleware for Wireless Sensor Networks." In A. Hinze and A. Buchmann, editors, *Principles and Apps. of Dist. Event-Based Systems*, chapter 9. IGI Global, 2010.

[Car08] P. Caramia, M. Dell'Olmo. "Multi-objective Management in Freight Logistics Increasing Capacity, Service Level and Safety with Optimization Algorithms." Springer, March 2008.

[CGV11] M. Chen, S. Gonzalez, A. V. Vasilakos, H. Cao, and V. C. Leung. "Body Area Networks: A Survey." *Mobile Netw. Appl.*, **16**(2), 2011.

[CJH10] Shuyi Chen, Kaustubh R. Joshi, Matti A. Hiltunen, Richard D. Schlichting, and William H. Sanders. "Blackbox prediction of the impact of DVFS on end-to-end performance of multitier systems." *ACM SIGMETRICS Performance Evaluation*, **37**(4), 2010.

[Coe98] C. A. Coello Coello. "Using the Min-Max Method to Solve Multiobjective Optimization Problems with Genetic Algorithms." In *Progress in Artificial Intelligence–IBERAMIA 98*. 1998.

[CRB11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya. "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms." *Software: Practice and Experience*, **41**(1):23–50, 2011.

[CSV12] Henri Casanova, Mark Stillwell, and Frédéric Vivien. "Virtual Machine Resource Allocation for Service Hosting on Heterogeneous Distributed Platforms." In *IEEE International Parallel & Distributed Processing Symposium*, May 2012.

[CWJ13] X. Chang, B. Wang, L. Jiqiang, W. Wang, and K. Muppala. "Green Cloud Virtual Network Provisioning Based Ant Colony Optimization." In *Proc. ACM Int'l Conference on Genetic and Evol. Computat*, 2013.

[CYH13] Wen-Yaw Chung, Pei-Shan Yu, and Chao-Jen Huang. "Cloud Computing System Based on Wireless Sensor Network." In *Federated Conference on Computer Science and Information Systems*, 2013.

[DAP00] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II." In *Proc. Conf. Parallel Problem Solving from Nature*, 2000.

[DDL07] N. Doulamis, A. Doulamis, A. Litke, A. Panagakis, T. Varvarigou, and E. Varvarigos. "Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing." *Elsevier Computer Comm.*, **30(3)**, 2007.

[Deb01] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons Inc, 2001.

[DLO14] Mianxiong Dong, He Li, K. Ota, L.T. Yang, and Haojin Zhu. "Multicloud-Based Evacuation Services for Emergency Management." *Cloud Computing, IEEE*, **1**(4):50–59, Nov 2014.

[DNA10] J.J. Durillo, A.J. Nebro, and E. Alba. "The jMetal Framework for Multi-Objective Optimization: Design and Architecture." In *Proc. of IEEE Congress on Evolutionary Computation*, 2010.

[DPA02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *IEEE Trans Evol. Computat.*, **6**(2), 2002.

[Eib02] A.E. Eiben. "Evolutionary Computing: the Most Powerful Problem Solver in the Universe." *Dutch Mathematical Archive*, **5**(3):126–131, 2002.

[FPP14] G. Fortino, D. Parisi, V. Pirrone, and G. Di Fatta. "BodyCloud: A SaaS Approach for Community Body Sensor Networks." *Future Generation Computer Systems*, **35**(6):62–79, 2014.

[GGQ13] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing." *J. Computer and System Sciences*, **79**(8), 2013.

[GKB11] Saurabh Kumar Garg, Pramod Konugurthi, and Rajkumar Buyya. "A linear programming-driven genetic algorithm for meta-scheduling on utility grids." *Int'l J. of Parallel, Emergent and Distributed Systems*, **26**(6):493–517, 2011.

[GLL09] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers." In *Proc. of ACM SIGCOM*, 2009.

[GMC13] Tom Guerout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru. "Energy-aware simulation with DVFS." *Simulation Modelling Practice and Theory*, **39**:96–91, 2013.

[GN07] P. Garbacki and V.K. Naik. "A Hybrid Linear Programming and Evolutionary Algorithm based Approach for On-line Resource Matching in Grid Environments." In *IEEE Int'l Conference on Cluster Computing and the Grid*, 2007.

[GP13] Hadi Goudarzi and Massoud Pedram. "Energy-Efficient Virtual Machine Replication and Placement in a Cloud Computing System." In *6th IEEE International Conference on Cloud Computing*, June 2013.

[GWM04] M. Gaynor, M. Welsh, S. Moulton, A. Rowan, E. LaCombe, and J. Wynne. "Integrating Wireless Sensor Networks with the Grid." *IEEE Internet Computing*, July/August 2004.

[GWT08] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, and S. Lu. "DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers." In *Proc. of ACM SIGCOM*, 2008.

[HSH09] Mohammad Mehedi Hassan, Biao Song, and Eui-Nam Huh. "A framework of sensor-cloud integration opportunities and challenges." In *Proc. the 3rd ACM Int'l Conference on Ubiquitous Info. Mgt. and Comm.*, 2009.

[Jon] Torna Omar Soro Junichi Suzuki Tadashi Nakano Jonathan S. Mitzman, Bria Morgan. "A feedback-based molecular communication protocol for noisy intrabody environments." In *E-health Networking, Application and Services (HealthCom), 2015 17th International Conference on*.

[KA09a] S. Khan and I. Ahmad. "A Pure Nash Equilibrium Based Game Theoretical Method for Data Replication Across Multiple Servers." *IEEE Transaction on Knowledge and Data Engineering*, **21**(4), 2009.

[KA09b] S. U. Khan and C. Ardil. "Energy Efficient Resource Allocation in Distributed Computing Systems." In *Proc. of WASET Int'l Conference on Distributed, High-Performance and Grid Computing*, 2009.

[KBK13] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. "DENS: data center energy-efficient network-aware scheduling." *Cluster Computing*, **16**(1), 2013.

[KK06] S. Kapadia and B. Krishnamachari. "Comparative Analysis of Push-Pull Query Strategies for Wireless Sensor Networks." In *Proc. International Conference on Distributed Computing in Sensor Systems*, 2006.

[KL03] M. Kodialam and T.V. Lakshman. "Detecting Network Intrusions via Sampling: A Game Theoretic Approach." In *Proc. of IEEE Conference on Computer and Communications Societies*, 2003.

[KOD14] T. Kumrai, Kaoru Ota, Mianxiong Dong, and P. Champrasert. "An incentive-based evolutionary algorithm for participatory sensing." In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 5021–5025, Dec 2014.

[Kun99] S. Kundu. "A Note on Optimizality vs. Stability - A Genetic Algorithm Based Approach." In *Proc. of World Congress on Structural and Multidisciplinary Optimization*, 1999.

[LGS06] M. Li, D. Ganesan, and P. Shenoy. "PRESTO: Feedback-Driven Data Management in Sensor Networks." In *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2006.

[LP07] W. B. Langdon and R. Poli. "Evolving Problems to Learn About Particle Swarm Optimizers and Other Search Algorithms." *IEEE Transactions on Evolutionary Computation*, **11**(5):561–578, 2007.

[LQL13] Xin Lia, Zhuzhong Qiana, Sanglu Lua, and Jie Wu. "Energy efficient virtual machine placement algorithm withbalanced and improved resource utilization in a data center." *Mathematical and Computer Modelling*, **58**(5-6), 2013.

[LWY09] von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. "Power-aware scheduling of virtual machines in DVFS-enabled clusters." In *IEEE International Conference on Clusters*, September 2009.

[LZW08] X. Li, J. Zhuang, S. Wang, and Y. Zhang. "A Particle Swarm Optimization Algorithm Based on Adaptive Periodic Mutation." In *Proc. of IEEE Int'l Conference on Natural Computation*, 2008.

[MF04] R. Masuchun and W.G. Ferrell. "Dynamic Rescheduling with Stability." In *Proc. of IEEE Asian Control Conference*, 2004.

[MLL12] Fei Ma, Feng Liu, and Zhen Liu. "Multi-objective Optimization for Initial Virtual Machine Placement in Cloud Data Center." *J. Infor. and Computational Science*, **9**(16), 2012.

[MW96] M.A. Marra and B.L. Walcott. "Stability and Optimality in Genetic Algorithm Controllers." In *Proc. of IEEE Int'l Symposium on Intelligent Control*, 1996.

[Now06] M. A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life*. Harvard University Press, 2006.

[NRT07] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[Per96] R. Perline. "Zipf's law, the central limit theorem, and the random division of the unit interval." *Physical Review E*, **54(1)**, 1996.

[PKG08] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. "Tiny web services: design and implementation of interoperable and evolvable sensor networks." In *Proc. the 6th ACM Int'l Conference on Embedded Network Sensor Systems*, 2008.

[PPB12] Shyamal Patel, Hyung Park, Paolo Bonato, Leighton Chan, and Mary Rodgers. "A Review of Wearable Sensors and Systems with Application in Rehabilitation." *Journal of Neuroengineering and Rehabilitation*, **9**(21), 2012.

[PS05] Joseph A. Paradiso and Thad Starner. "Energy Scavenging for Mobile and Wireless Electronics." *Pervasive Computing*, **4**(1):1827, 2005.

[PSO14] D. H. Phan, J. Suzuki, S. Omura, K. Oba, and A. Vasilakos. "Multiobjective Communication Optimization for Cloud-integrated Body Sensor Networks." In *Proc. IEEE/ACM Int'l Workshop on Data-intensive Process Management in Large-Scale Sensor Systems: From Sensor Networks to Sensor Clouds, In conjunction with IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing*, 2014.

[RAS] Barry Rountree, Dong Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. "Beyond DVFS: A first look at performance under a hardware-enforced power bound." In *Proc. 8th Workshop on High-Performance, Power-Aware Computing*.

[RKW10] Carlos Oberdan Rolim, Fernando Luiz Koch, Carlos Becker Westphall, Jorge Werner, Armando Fracalossi, and Giovanni Schmitt Salvador. "A Cloud Computing Solution for Patient's Data Collection in Health Care Institutions." In *Proc. the 2nd IARIA Int'l Conference on eHealth, Telemedicine and Social Medicine*, 2010.

[Sal10] David Salerno. "Ultralow Voltage Energy Harvester Uses Thermoelectric Generator for Battery-Free Wireless Sensors." *LT Journal of Analog Innovation*, **20**(3), 2010.

[SD95] N. Srinivas and K. Deb. "Multiobjective function optimization using nondominated sorting genetic algorithms." *Evol. Computat.*, **2**(3), 1995.

[SH06] T. C. Shan and W. W. Hua. "Solution Architecture for N-Tier Applications." In *Proc. of IEEE Int'l Conference on Services Computing*, September 2006.

[SQM12] N.K. Suryadevara, M. T. Quazi, and S.C. Mukhopadhyay. "Intelligent Sensing Systems for Measuring Wellness Indices of the Daily Activities for the Elderly." In *8th Int'l Conference on Intelligent Environments*, 2012.

[SZL08] R. Subrata, A. Y. Zomaya, and B. Landfeldt. "Game Theoretic Approach for Load Balancing in Computational Grids." *IEEE Trans. Parallel and Distributed Systems*, **19**(1), 2008.

[TD08] V. Toğan and A.T. Daloğlu. "An Improved Genetic Algorithm with Initial Population Strategy and Self-Adaptive Member Grouping." *Computers and Structures*, **86**(11-12):1204–1218, 2008.

[TEC08] H. A. Taboada, J. F. Espiritu, and D. W. Coit. "MOMS-GA: A Multi-Objective Multi-State Genetic Algorithm for System Reliability Optimization Design Problems." *IEEE Transactions on Reliability*, **57**(1), 2008.

[THL03] J. Tatemura, W-P. Hsiung, and W-S. Li. "Acceleration of Web Service Workflow Execution through Edge Computing." In *Proc. of Int'l WWW Conference*, 2003.

[TJ78] P. Taylor and L. Jonker. "Evolutionary stable strategies and game dynamics." *Elsevier Mathematical Biosciences*, **40(1)**, 1978.

[UPS05] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. "An analytical model for multi-tier internet services and its applications." In *Proc. of ACM Int'l Conference on Measurement and Modeling of Computer Systems*, June 2005.

[VT99] C. Voudouris and E.P.K. Tsang. "Guided Local Search and its application to the Travelling Salesman Problem." **113**(2):469–499, 1999.

[WBS10] H. Wada, P. Boonma, and J. Suzuki. "Chronus: A Spatiotemporal Macroprogramming Language for Autonomic Wireless Sensor Networks." In N. Agoulmine, editor, *Autonomic Network Mgt. Principles: From Concepts to Applications*, chapter 8. Elsevier, 2010.

[Wei96] J. W. Weibull. *Evolutionary Game Theory*. MIT Press, 1996.

[WKL10] Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Chien An Lai, Masazumi Matsubara, and Calton Pu. "Impact of DVFS on n-Tier Application Performance." In *In Proc. of ACM Conference on Timely Results in Operating Systems*, November 2010.

[WSY12] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. "E3: A Multiobjective Optimization Framework for SLA-aware Service Composition." *IEEE Trans. Services Computing*, **5**(3), 2012.

[WVZ09] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. "A game-theoretic method of fair resource allocation for cloud computing services." *J. Supercomputing*, **54**(2), 2009.

[XHT11] Yi Xu, Sumi Helal, My Thai, and Mark Scmalz. "Optimizing push/pull envelopes for energy-efficient cloud-sensor systems." In *Proc. the 14th ACM Int'l Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2011.

[YK10] M. Yuriyama and T. Kushida. "Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing." In *Proc. the 13th Int'l Conf. on Network-Based Info. Sys.*, 2010.

[YKK08] O. Yugay, I. Kim, B. Kim, and F.I.S. Ko. "Hybrid Genetic Algorithm for Solving Traveling Salesman Problem with Sorted Population." In *Proc. of IEEE Int'l Conference on Convergence and Hybrid Information Technology*, 2008.

[ZT98] E. Zitzler and L. Thiele. "Multiobjective Optimization Using Evolutionary Algorithms: A Comparative Study." In *Proc. Int'l Conf. on Parallel Problem Solving from Nature*, 1998.

[ZYS13] Peng Zhang, Zheng Yan, and Hanlin Sun. "A Novel Architecture Based on Cloud Computing for Wireless Sensor Network." In *2nd International Conference on Computer Science and Electronics Engineering*, 2013.