E³: Multi-Objective Genetic Algorithms for SLA-aware Service Deployment Optimization Problem

Article *in* IEEE Transactions on Services Computing · January 2012

DOI: 10.1109/TSC.2011.6 · Source: IEEE Xplore

4 authors, including:

Junichi Suzuki
University of Massachusetts Boston
163 PUBLICATIONS **1,701** CITATIONS

E^3: A Multiobjective Optimization Framework for SLA-aware Service Composition

Hiroshi Wada, Junichi Suzuki, Yuji Yamano and Katsuya Oba

Abstract—In Service Oriented Architecture, each application is often designed as a set of services. Each service is deployed as one or more service instances that operate at different quality of service (QoS) levels. When a number of instances are available for each service, application developers are required to determine an appropriate composition of service instances that satisfies a given service level agreement (SLA), which defines a set of QoS constraints for each application. This problem, the SLA-aware service composition (SSC) problem, is known NP-hard, which can take a significant amount of time and costs to find the optimal solutions (i.e., the optimal combinations of service instances) from a huge number of solution candidates. This paper proposes an optimization framework, called E^3, to solve the SSC problem. E^3 provides multiobjective genetic algorithms that heuristically seek the Pareto-optimal solutions with respect to QoS objectives under given SLAs. Simulation results demonstrate that E^3 finds quality solutions in a reasonably short time and E^3 consistently outperforms a well-known existing multiobjective genetic algorithm.

Index Terms—Optimization of services composition, Quality of service, Service-level agreements, Multiobjective genetic algorithms

1 Introduction

Service Oriented Architecture (SOA) is an emerging style of software architectures that reuses and combines loosely-coupled services for building, maintaining and integrating applications in order to improve productivity and cost effectiveness throughout application lifecycle [1], [2]. In SOA, each application is often designed with a set of *service*s and a *workflow* (or business process). Each service encapsulates the function of an application component. Each workflow defines how services interact with each other.

When a service oriented application operates, it is instantiated as a workflow instance that deploys each service in the application as one or more service instances. Each service instance follows a particular deployment plan; different service instances operate at different quality of service (QoS) levels. For example, Amazon Elastic Compute Cloud (EC2)[1] provides eight different deployment plans (called *EC2 instance*s) that yield different QoS levels for service instances by provisioning different amounts of resources. When an application is intended to serve different categories of users (e.g., users with for-fee and free memberships), it is instantiated with multiple workflow instances, each of which is responsible for offering a specific QoS level to a particular user category.

In SOA, a service level agreement (SLA) is defined upon a workflow instance as its end-to-end QoS requirements such as throughput, latency and cost (e.g., resource utilization fees) for each user category. In order to satisfy given SLAs, application developers are required to optimize a composition of service instances, *service composition*, for each user category by considering which service instances to be used for each service and how many service instances to be used for each service. For example, a service composition may be intended to improve the latency of a heavily-accessed service by deploying it as an expensive service instance that allocates a large amount of resources. Another service composition may deploy a service as two inexpensive service instances connected in parallel for improving the service's throughput.

This decision-making problem, called the SLA-aware service composition (SSC) problem, is a combinatorial optimization problem that searches the optimal bindings between each service and its service instances. This problem has two key research issues. First, it is known NP-hard [3], which can take a significant amount of time, labor and costs to find the optimal service compositions (i.e., optimal combinations of service instances) from a huge search space (i.e., a huge number of possible service instance combinations). The second issue is that the SSC problem often faces tradeoffs among conflicting QoS objectives in SLAs. Therefore, there exits no single optimal solution but rather a set of alternative solutions of equivalent quality, which is called Pareto-optimal solutions. For example, in order to reduce its latency, a service may be bound to an expensive resource-rich service instance; however, this is against another objective to reduce cost. Minimizing latency and cost clearly conflicts with each other; therefore, there is no single optimum to be found with respect to the latency and cost objectives.

Given a set of Pareto-optimal solutions, application developers can examine the trade-offs among different service compositions with respect to different QoS objectives. For example, they can compare the service compositions yielding low throughput with a low cost, high throughput with a high cost and intermediate throughput with an intermediate cost,

- *H. Wada was with University of Massachusetts Boston, Boston, MA 02125 USA. He is currently with National ICT Australia, Eveleigh, NSW 1430 Australia, and School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052 Australia (e-mail: hiroshi.wada@nicta.com.au).*
- *J. Suzuki is with the Department of Computer Science, University of Massachusetts Boston, Boston, MA 02125, USA (e-mail: jxs@cs.umb.edu).*
- *Y. Yamano and K. Oba are with OGIS International, Inc., San Mateo, CA 94404, USA (e-mail: yyamano, oba@ogis-international.com).*

Manuscript received xx xx, xxxx: revised xx xx, xxxx.

[1] www.amazon.com/ec2

and decide which one they use for their applications based on their preferences and priorities.

Although a large number of research work (e.g., [4]–[9]) have leveraged linear programming to solve the SSC problem, they suffer from high computational costs. (It does not scale well as the problem's complexity grows.) Also they do not reveal multiple Pareto-optimal service compositions because linear programming is designed to find a single optimal solution. To address the issue of high computational costs, several heuristics have been studied (e.g., [3], [10]–[16]); however, most of them are not designed to search Pareto-optimal solutions.

This paper investigates an optimization framework, called E^3 (Evolutionary multiobjective sErvice composition optimizEr), to solve the SSC problem. E^3 defines a service composition model and provides two multiobjective genetic algorithms: E^3-MOGA and Extreme-E^3 (X-E^3). Both are multiobjective genetic algorithms (GAs) that heuristically search Pareto-optimal service compositions that satisfy given SLAs. E^3-MOGA is designed to search Pareto-optimal solutions that are equally distributed in the objective space. Therefore, it can produce both extreme service compositions (e.g., the one yielding high throughput with a high cost) and balanced service compositions (e.g., the one yielding intimidate throughput with an intermediate cost) at the same time. Given multiple Pareto-optimal service compositions, each application developer can understand the trade-offs among them and make a well-informed decision to choose one of them, as the best service composition, according to his/her preferences and priorities.

X-E^3 is designed to search extreme service compositions. It aids to reveal the maximum range of QoS trade-offs that an application can yield. For example, X-E^3 can reveal the maximum range of trade-offs between latency and cost. This allows developers to make well-informed decisions on whether they may want to spend more to gain lower latency or accept higher latency to save expenditure.

This paper is organized as follows. Section 2 shows the service composition model and QoS aggregation model in E^3. Sections 3 and 4 describe the details of E^3-MOGA and X-E^3, respectively. Section 5 presents a series of simulation results to evaluate E^3-MOGA and X-E^3. Sections 6 and 7 conclude with some discussion on related work.

2 Service Composition and QoS Aggregation Models in E^3

E^3 assumes that a workflow consists of a set of abstract services (Table 1). Figure 1 shows an example workflow, which is modeled with a well-known service oriented application for loan processing and brokerage [17]. The workflow consists of four abstract services: Loan Validator Service, Credit Rating Service, Loan Broker Service and Loan Sale Processor Service. A loan validator service first receives a loan application as a message from an applicant and examines whether it is complete. If it is complete, a credit rating service retrieves the applicant's credit information. A loan broker service also examines the application for sale to a secondary loan company and lists appropriate ones for the applicant. Finally, the application is forwarded to a loan sale processor service. Each abstract service defines the interface of a certain function such as application validation and credit information retrieval; however, it provides no specific implementation of the function (Table 1).

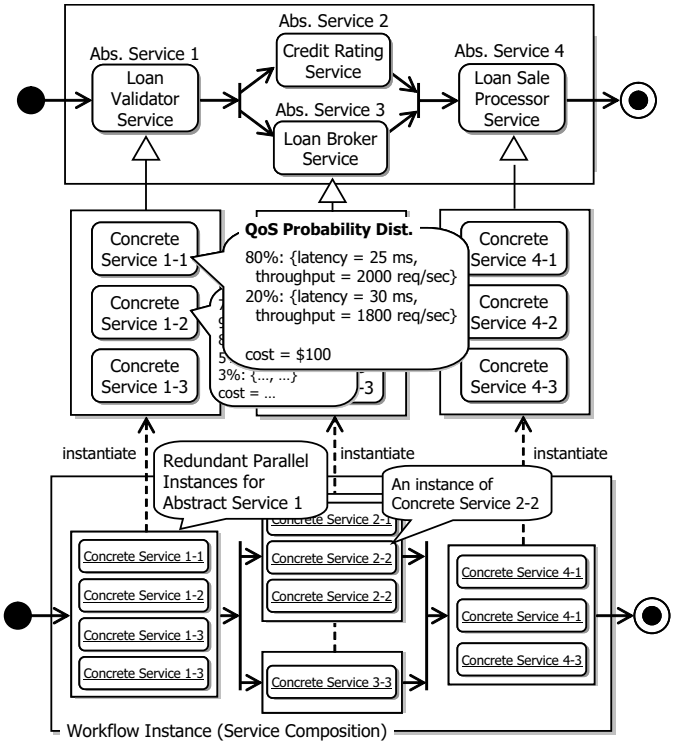

Fig. 1: Service Composition Model in E^3

TABLE 1: Elements in the Service Composition Model in E^3

Workflow	A series of abstract services to execute to fulfill an application's function.
Abstract Service	Interface and functional definition of an application component.
Concrete Service	An implementation and realization of an abstract service with a particular deployment plan.
Service Instance	An instance of a concrete service
Workflow Instance	An instance of a workflow. Responsible for each user category. A composition of service instances bound to abstract services.

Each concrete service implements an abstract service and realizes it with a particular deployment plan that is characterized with three QoS attributes: throughput, latency and cost. For example, a concrete service may implement Loan Validator Service as a RESTful service on a LAMP stack[2] and realize it with a deployment plan in Amazon EC2. As of this

[2]LAMP is an open-source solution stack of Linux, Apache HTTP server, MySQL and Perl/PHP/Python.

writing, Amazon EC2 offers a deployment plan with a 1.0 GHz CPU and 1.7 GB memory for $0.095 per hour in Northern California. Another deployment plan with four 2.0 GHz CPU cores and 15.0 GB memory costs $0.76 per hour. Depending on deployment plans, different concrete services yield different QoS measures while they offer the identical function (Table 1).

E^3 assumes that QoS probability distribution is known for each concrete service. For example, it is obtained based on QoS history of concrete services. In Figure 1, Concrete Service 1-1 yields the throughput of 2,000 request messages per second and the latency of 25 milliseconds at the probability of 80%. Its deployment cost is $100 a month.

Service instances instantiate concrete services and form a workflow instance. In a workflow instance, at least one service instance must be bound to each abstract service. Multiple service instances can be bound to an abstract service in parallel for improving the service's throughput and fault tolerance. (They are called *redundant parallel* instances.) In Figure 1, four redundant parallel instances are bound to Loan Validator Service. An SLA is defined with the end-to-end throughput, latency and cost of a workflow instance on a per user category basis (Table 1).

Given a workflow definition and a set of concrete services, E^3 seeks the Pareto-optimal service compositions (i.e. workflow instances) that satisfy certain SLAs by considering how many instances of each concrete service to use and how to structure those service instances in a workflow instance. For judging whether a service composition satisfies a given SLA, E^3 examines its end-to-end QoS by aggregating QoS measures of individual service instances. First, E^3 calculates the expected QoS of each service instance as $E(X) = \sum x_i p_{x_i}$ where $X = (x_0, ..., x_n)$ denotes a set of possible QoS measures and $p_{x_i} = P\{X = x_i\}$ denotes their corresponding probabilities ($\sum p_{x_i} = 1$). For example, the expected throughput of an instance of Concrete Service 1-1 is 1,960 requests/second $(2{,}000 \times 0{,}8 + 1{,}800 \times 0.2)$.

Once the expected QoS of each service instance is obtained, E^3 computes each abstract service's QoS by applying aggregate functions (Table 2) to service instances bound to the abstract service. In this computation, each abstract service is interpreted as a collection of service instances in redundant parallel. In an example workflow instance shown in Figure 2, the expected QoS of Abstract Service 1 is computed by aggregating the QoS values of five service instances in redundant parallel. E^3 assumes that service instances in redundant parallel are used in equal probability. Thus, the aggregated throughput is the summation of throughput measures that service instances yield. The aggregated latency is the average of latency measures that service instances yield. The aggregated cost is the summation of costs that service instances generate (Table 2).

When abstract services are connected in parallel, as Abstract Services 2 and 3 in Figure 2, parallel execution flows are joined (synchronized) at the end. Therefore, the aggregated throughput of abstract services in parallel is computed as the minimal throughput in parallel execution flows. The aggregated latency is computed as the maximum latency in parallel flows. The aggregated cost is the summation of costs generated by individual flows (Table 2).

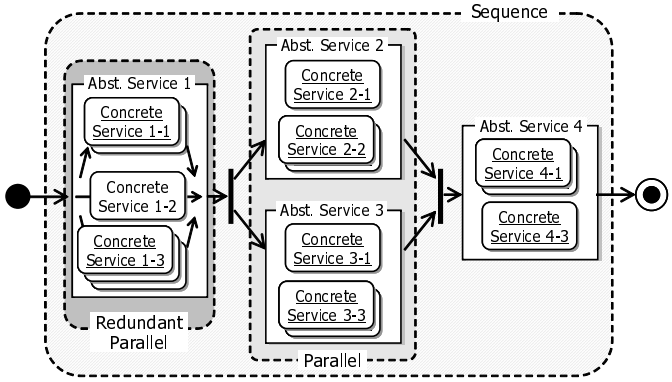

Fig. 2: An Example Workflow Instance

In a sequence of abstract services, abstract services are executed sequentially one by one. Therefore, the aggregated throughput of sequential abstract services is computed as the minimal throughput that abstract services yield in a sequence. The aggregated latency is computed as the summation of latency measures that abstract services yield. The aggregated cost is computed as the summation of costs that abstract services generate (Table 2).

Any workflow instance can be decomposed to a collection of redundant parallel, parallel and sequence structures of service instances. Thus, E^3 computes the end-to-end QoS of each workflow instance by recursively applying aggregate functions in Table 2.

3 Multiobjective Optimization of Service Composition with E^3-MOGA

This section describes the design of E^3-MOGA.

3.1 Individual Representation and Optimization Objectives

E^3-MOGA maintains a population of individuals, each of which represents service compositions (i.e., workflow instances) for different user categories. Currently, E^3-MOGA assumes three user categories: platinum, gold and silver. Thus, each individual contains three service compositions, each of which is responsible for a user category. E^3-MOGA evolves and optimizes individuals through generations by repeatedly applying genetic operations (selection, reproduction, crossover and mutation) to them in each generation.

Figure 3 shows an example individual that consists of 36 values. Each value denotes the number of instances to be deployed for each concrete service. In this example, a workflow consists of four abstract services, and each of them has three concrete services. Thus, 12 (4×3) values are used to represent a service composition for a user category. Considering three user categories, each individual represents three service compositions for platinum, gold and silver users with 36 values (12×3). An individual in Figure 3 represents a service composition that has two instances for Concrete Service 1-1 and two instances for Concrete Service 3-3 for

TABLE 2: Aggregate Functions

QoS Attribute	Redundant Parallel	Parallel	Sequence
Throughtput (T)	$\sum_{i \in service\ instances} T_i$	$Min_{a \in abstract\ services} T_a$	$Min_{a \in abstract\ services} T_a$
Latency (L)	$Avg_{i \in service\ instances} L_i$	$Max_{a \in abstract\ services} L_a$	$\sum_{a \in abstract\ services} L_a$
Cost (C)	$\sum_{i \in service\ instances} C_i$	$\sum_{a \in abstract\ services} C_a$	$\sum_{a \in abstract\ services} C_a$

platinum users. Individuals are evaluated and optimized with respect to 10 objectives described in Figure 3.

X-E^3 use the same individual representation and optimization objectives that E^3-MOGA uses.

3.2 Optimization Process in E^3

Listing 1 shows the optimization process in E^3-MOGA. At each (the g-th) generation, two parents, p_α and p_β, are selected from the current population (P^g) with binary tournaments [18]. They reproduce two offspring by performing one-point crossover with `Crossover()`. (A crossover point is randomly selected.) Then, the offspring are randomly altered with `Mutation()`. Mutation occurs on values in each individual at the mutation rate of $1/n$ where n denotes the number of values in an individual. This means that, probabilistically, one value is mutated in each offspring. This reproduction process is repeated with parent selection, crossover and mutation operations until the number of offspring (Q^g) reaches μ. From a pool of $P^g \cup Q^g$, E^3-MOGA selects the top μ individuals based on their fitness values. A fitness value indicates a quality (or goodness) of an individual; it is calculated with a fitness function in `AssignFitnessToIndividuals()`. E^3 repeats the above process for g_{max} times.

Listing 1: E^3-MOGA

```
g ← 0
P^0 ← Randomly generated μ individuals
repeat until g == g_max {
  AssignFitnessValues(P^g)
  Q^g ← ∅
  repeat until |Q^g| == μ {
    // Parent selection via binary tournament
    p_a, p_b ← RandomSelection(P^g)
    p_α ← BTSelection(p_a, p_b)
    p_a, p_b ← RandomSelection(P^g)
    p_β ← BTSelection(p_a, p_b)

    // Reproduction via one-point crossover
    q_1, q_2 ← Crossover(p_α, p_β)

    // Mutation on reproduced offspring
    q_1 ← Mutation(q_1)
    Add q_1 to Q^g if Q^g does not contain q_1.
    q_2 ← Mutation(q_2)
    Add q_2 to Q^g if Q^g does not contain q_2.
  }
  AssignFitnessValues(P^g ∪ Q^g)
  P^{g+1} ← Top μ of P^g ∪ Q^g
  g ← g+1
}

AssignFitnessValues(P){
  DominationRanking(P)
  foreach p in P {
    if p is feasible
      // Fitness function for a feasible individual
      f ← p's domination value ×
          p's distance from the worst point ×
          p's sparsity
    else
      // for an infeasible individual
      f ← 0 - p's violation / p's domination value
    p's fitness value ← f
  }
}
```

TABLE 3: Variables and Functions in E^3-MOGA

g_{max}	Maximum number of generations
μ	Population size
P^g	A set of individuals at the g-th generation
Q^g	A set of offspring generated at g-th generation
`RamdonSelection(`P`)`	A function returning two randomly selected individuals from P
`BTSelection(`p_1`, `p_2`)`	A function returning either p_1 or p_2 that has a higher fitness value
`Corssover(`p_1`, `p_2`)`	A function returning two individuals created by one point crossover between p_1 and p_2
`Mutation(`p_k`)`	A function randomly changing p_k's values with $1/n$ mutation rate
`DominationRanking(`P`)`	A function ranking individuals in the population P based on their constraint-domination relationships

E^3 is designed to seek individuals that satisfy given SLAs and exhibit the optimal trade-offs among QoS objectives in the SLAs. In order to fulfill the both requirements, E^3 distinguishes *feasible* individuals, which satisfy SLAs, and *infeasible* individuals, which do not. E^3 uses two different fitness functions for feasible and infeasible individuals. (See `AssignFitnessToIndividuals()`.) The fitness function for feasible individuals is designed to encourage them to improve their QoS values in 10 objectives and maintain diversity in their QoS values. The fitness function for infeasible individuals is designed to encourage them to reduce SLA violations and turn into feasible individuals. Feasible individuals have positive fitness values, while infeasible ones have negative fitness values. E^3 considers higher-fitness individuals as higher quality (or better). Therefore, feasible individuals have higher chances, than infeasible ones, to be selected as parents for reproduction.

3.3 Domination Value

In the design of fitness functions, E^3 employs the notion of *domination value* [19]. A domination value is determined based on the notion of *domination rank*, and a domination rank is determined based on the notion of *constraint-domination* [20]. An individual i is said to *constraint-dominate* an individual j when any of the following conditions are hold.

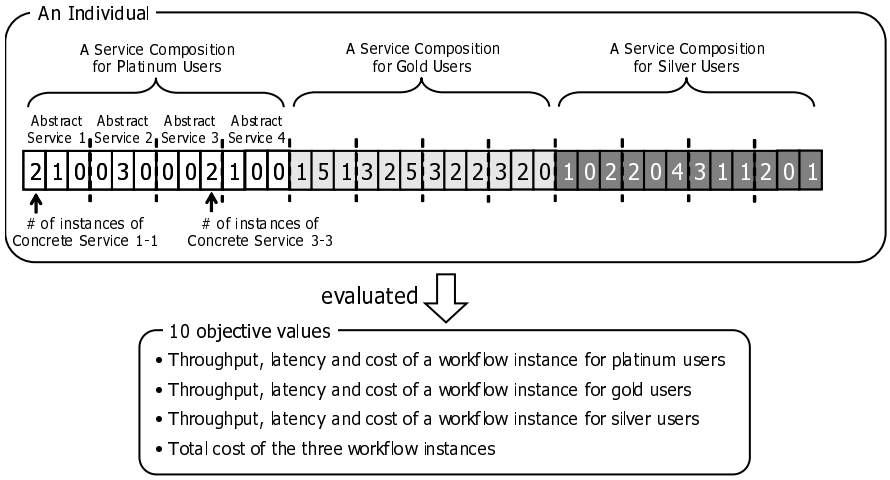

Fig. 3: An Example Individual and Optimization Objectives

1) Individual i is feasible and j is not.
2) Both i and j are feasible, and i dominates j in terms of their QoS values.
3) Both i and j are infeasible, and i dominates j in terms of their SLA violations.

In the second condition, an individual i is said to *dominate* an individual j when both of the following conditions are hold.

$$\{\forall\left(o_{max}^{i,k}, o_{max}^{j,k}\right) | o_{max}^{i,k} \geq o_{max}^{j,k}\} \text{ and } \{\forall\left(o_{min}^{i,k}, o_{min}^{j,k}\right) | o_{min}^{i,k} \leq o_{min}^{j,k}\}. \quad (1)$$

$$\{\exists\left(o_{max}^{i,k}, o_{max}^{j,k}\right) | o_{max}^{i,k} > o_{max}^{j,k}\} \text{ or } \{\exists\left(o_{min}^{i,k}, o_{min}^{j,k}\right) | o_{min}^{i,k} < o_{min}^{j,k}\}. \quad (2)$$

where $o_{max}^{i,k}$ and $o_{min}^{i,k}$ are individual i's k-th objective value (QoS value) to be maximized and minimized, respectively.

In the third condition, an individual i is said to *dominate* an individual j when both of the following conditions are hold.

$$\{\forall\left(v^{i,k}, v^{j,k}\right) | v^{i,k} \leq v^{j,k}\}. \quad (3)$$

$$\{\exists\left(v^{i,k}, v^{j,k}\right) | v^{i,k} < v^{j,k}\}. \quad (4)$$

where $v^{i,k}$ is individual i's SLA violation in terms of k-th objective.

Figure 4 shows an example of the second condition. It examines domination relations among six feasible individuals in terms of their QoS values. For simplicity, this figure uses two dimensional objective spaces with two of 10 QoS objectives. In this example, individuals A, B and C are at the first rank. In other words, they are *non-dominated*. A, B and C do not dominate with each other because each of them cannot dominate the others. Since individuals D and E are dominated by the top rank individuals, they are at the second rank. Similarly, individual F is at the third rank. A domination rank is assigned to an individual based on the number of individuals in the same or lower domination ranks than the individual in question. Therefore, non-dominated individuals have the highest domination values (Figure 4).

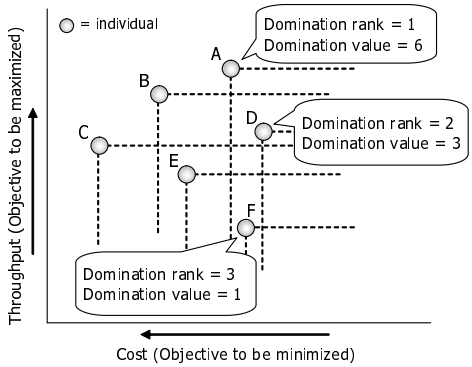

Fig. 4: An Example Domination Ranking

3.4 Fitness Function for Feasible Individuals

For each feasible individual, E^3-MOGA assigns the product of a domination value, *distance from the worst point* and *sparsity* as a fitness value.

It is known that most individuals in a population tend to become non-dominated when multiobjective problems have more than three objectives, and it weakens the pressure to evolve individuals since most solutions are in the same domination rank (e.g., all solutions have the save domination value) [21]. In order to avoid this issue, E^3-MOGA uses the distance from the worst point, i.e., a point consists of the worst objective values in a population, to measure the excellence of individuals as well as domination values. For each feasible individual E^3-MOGA calculates the Manhattan distance from the worst point. (In this calculation, QoS values are normalized in case QoS objectives have different scales.) Manhattan distance is used because it becomes larger, compared with the other p-norm distances ($p \geq 2$) including Euclidian distance ($p = 2$), when individuals' objective values are balanced.

Sparsity represents diversity of individuals; how individuals spread uniformly in the objective space. Without taking preventive measures, individuals in a population tend to form relatively few clusters and cannot yield a whole set of potential solutions. Therefore, in order to reveal wider variety of trade-offs in objectives. Maintaining the diversity of individuals is an important consideration in E^3-MOGA. For each individual E^3 calculates the Manhattan distance to the closest neighbor individual in the objective space and determines the distance as its sparsity. (In this calculation, QoS values are normalized

in case QoS objectives have different scales.) Manhattan distance is used because it becomes larger, when individuals are uniformly distributed over all objectives. E^3 favors diversity of individuals because diverse individuals can reveal a wide range of trade-offs among QoS objectives.

3.5 Fitness Function for Infeasible Individuals

For each infeasible individual E^3 assigns a fitness value based on its *SLA violations* and domination value using the following equation.

$$f = \sum \frac{v^k}{\text{domination value}}$$

where v^k is an SLA violation in terms of k-th objective. (SLA violations are normalized in case QoS requirements have different scales.)

SLA violations contribute to decrease a fitness value, while a domination value contributes to increase it.

4 Multiobjective Optimization of Service Composition with X-E^3

The purposes of X-E^3 is different from that of E^3-MOGA, i.e., X-E^3 aims to seek extreme solutions while E^3 seeks solutions equally distributed over the objective space. However, X-E^3 and E^3-MOGA shares many design details such as individual representation, the optimization process and genetic operations. The only difference between them is the design of a fitness function for feasible individuals. For each feasible individual X-E^3 calculates the *degree of extremeness* and assign it as a fitness value.

X-E^3 favors feasible individuals that have extreme objective values in a couple of objectives even if they are dominated by other individuals. For each individual X-E^3 calculates the standard score of each objective value. When an individual has 10 different objectives, the individual has 10 different standard scores. The standard score z_i^k, individual i's standard score in terms of k-th objective, is calculated as follows.

$$z_i^k = \begin{cases} (x_i^k - \mu^k)/\sigma^k & k\text{-th objective is to be maximized} \\ (\mu^k - x_i^k)/\sigma^k & k\text{-th objective is to be minimized} \end{cases}$$

where x_i^k is individual i's objective value in terms of k-th objective, μ^k is the average of k-th objective values of a population, and σ^k is the standard deviation of k-th objective values of a population.

Then, individual i's fitness value f_i is calculated as follows.

$$f_i = \sum_{p=1}^{n} \frac{1}{exp(p-1)} \times z_{i,p}$$

where $z_{i,p}$ is the p-th largest standard score of individual i. n is the number of objectives. The term $1/exp(p-1)$ works as a weight value that decreases with p. For example, a weight for the largest standard score (i.e., $p = 1$) is 1.0. A weight for the third largest standard score is 0.135. This way, X-E^3 weights a couple of extreme objective values and virtually ignores objective values that are similar to or worse than other individuals. This design strategy helps reveal extreme individuals rather than balanced ones.

5 Simulation Evaluation

This section investigates the performance characteristics of E^3-MOGA and X-E^3 through four simulation studies. The first study evaluates how E^3-MOGA and X-E^3 optimize service compositions and satisfy given SLAs. The second study examines the impacts of workflow complexity on E^3-MOGA and X-E^3. The third study evaluates the quality of solutions that E^3-MOGA and X-E^3 produce by comparing them with the real optimal solutions. The fourth study evaluates the quality of combined solutions that E^3-MOGA and X-E^3 produce. All simulation studies compare E^3-MOGA and X-E^3 with NSGA-II, which is one of the most well-known multiobjective genetic algorithms [20]. All simulation results were measured with a Sun Java SE 6.0 VM running on a Windows Server 2003 PC with an Intel Xeon 2.13 GHz Dual Core CPU and 4.0 GB memory space.

5.1 Deployment Optimization and SLA Satisfaction

This simulation study simulates a workflow that consists of four abstract services shown in Figure 5. Each abstract service is associated with three concrete services that operate at different QoS levels (Table 4): high-end (high throughput, low latency and high cost), low-end (low throughput, high latency and low cost) and intermediate (intermediate throughput, latency and cost). Table 5 shows the SLAs (i.e., QoS constraints) defined for three user categories. Platinum and gold users have throughput and latency constraints and have no cost (or budget) constraints. Silver users have throughput and cost constraints. In addition, there is a constraint on the total costs generated by all of three user categories. E^3-MOGA, X-E^3 and NSGA-II use the population size (μ) of 100 and the maximum generation (g_{max}) of 500. All results are the average of 100 independent simulation runs.

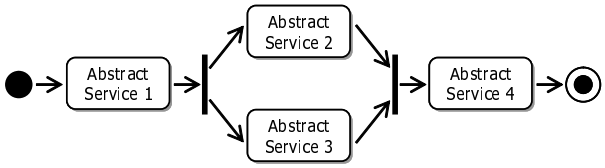

Fig. 5: A Workflow used in Simulations

Figures 6 to 15 shows 10 different QoS measures in 10 optimization objectives. Figures 6, 7 and 8 show the maximum throughput that feasible individuals yield for platinum, gold and silver users, respectively. The initial individuals are randomly generated, and all of them are infeasible at the beginning of a simulation. Thus, it takes approximately 20 generations to find feasible individuals, which satisfy SLAs in all user categories. After that, feasible individuals evolve to yield higher throughput over generations. Since X-E^3 is designed to find extreme QoS objective values, it produces individuals that yield higher throughput compared to E^3-MOGA and NSGA-II. E^3-MOGA's individuals tend to yield higher throughput compared to NSGA-II's individuals since E^3-MOGA is designed to find a wider variety of individuals.

Figures 9, 10 and 11 show the minimum latency that feasible individuals yield for three different user categories. Similar to

TABLE 4: QoS of Concrete Services

Abstract Service	Concrete Service	QoS Probability Distribution			Cost
		Prob. (%)	Throughput	Latency	
1	1	0.85	9,000	60	90
		0.05	10,000	50	
		0.05	6,000	80	
		0.05	0	0	
	2	0.80	5,500	60	50
		0.15	4,000	100	
		0.05	0	0	
	3	0.30	2,000	200	10
		0.30	3,000	180	
		0.20	1,500	250	
		0.20	0	0	
2	1	0.70	2,000	20	50
		0.30	2,300	18	
	2	0.90	4,000	15	100
		0.05	6,000	13	
		0.05	3,000	20	
	3	0.70	4,000	25	70
		0.20	3,000	23	
		0.05	2,500	30	
		0.05	0	0	
3	1	0.70	1,500	30	30
		0.30	2,000	20	
	2	0.80	3,000	12	80
		0.10	5,000	20	
		0.10	500	80	
	3	0.50	1,000	60	10
		0.30	500	50	
		0.20	0	0	
4	1	0.75	2,500	50	20
		0.25	3,000	55	
	2	0.90	6,000	15	70
		0.05	4,000	20	
		0.05	3,000	20	
	3	0.85	1,000	90	5
		0.05	500	120	
		0.05	100	150	
		0.05	0	0	

TABLE 5: SLAs (QoS Constraints)

Users	Constraints (Upper/Lower Bound)			
	Throughput (Lower)	Latency (Upper)	Cost (Upper)	Total Cost (Upper)
Platinum	12,000	100	-	2,000
Gold	6,000	130	-	
Silver	2,000	-	250	

the throughput results in Figures 6, 7 and 8, X-E^3 finds better (or more extreme) individuals compared to E^3-MOGA and NSGA-II. E^3-MOGA consistently outperforms NSGA-II.

Figures 12, 13 and 14 show the minimum cost that feasible individuals generate for three different user categories. As Table 5 illustrates, platinum and gold users do not have cost constraints. Also, platinum users have more demanding throughput and latency constraints than gold users. Thus, the service compositions for platinum users tend to have a number of high-end concrete service instances. As a result, platinum users incur higher costs than gold users. In Figure 12, E^3-MOGA outperforms X-E^3. This is because the extremeness in X-E^3 (i.e., standard score) tends to be small in the cost objectives since the range of cost values is not large (i.e., the search space is limited in the dimensions of costs). This weakens the pressure to evolve and optimize individuals with respect to costs.

Figure 15 shows the minimum total cost. X-E^3 outperforms E^3-MOGA and NSGA-II. E^3-MOGA converges faster than NSGA-II, although the two algorithms yield similar cost measures over generations.

Figures 6 to 15 demonstrate that E^3-MOGA and X-E^3 perform as they are designed and they yield comparable or superior results against NSGA-II in all 10 QoS objectives.

Figure 16 shows the number of feasible individuals over generations. E^3-MOGA and X-E^3 find feasible individuals faster than NSGA-II because they consider both domination values and the degree of SLA violation to evaluate infeasible individuals (Section 3.5), while NSGA-II simply uses domination ranks obtained through its constraint-domination ranking [20]. E^3-MOGA and X-E^3 use the same fitness function for evaluating infeasible individuals; therefore, the number of feasible individuals increase similarly with the two algorithms.

Figure 17 shows the number of feasible individuals over generations under more stringent SLAs (Table 6) than those in Table 5. More stringent SLAs make it harder to find feasible solutions. As Figure 17 illustrates, E^3-MOGA and X-E^3 evolve feasible individuals faster than NSGA-II. Compared with Figure 16, E^3-MOGA and X-E^3 exhibit higher superiority over NSGA-II under more stringent SLAs.

TABLE 6: Stringent SLAs (Stringent QoS Constraints)

Users	Constraints (Upper/Lower Bound)			
	Throughput (Lower)	Latency (Upper)	Cost (Upper)	Total Cost (Upper)
Platinum	14,000	95	-	1,800
Gold	8,000	130	-	
Silver	2,000	-	250	

Figures 18 to 21 show the distribution and coverage of individuals in the objective space. Figure 18 shows the average coefficient of variation (CV) of QoS objective values. As a dimensionless and scale invariant variance, CV is defined as: $CV = \sigma/\mu$ where σ and μ are the standard deviation and the mean of QoS values, respectively. A lower CV means that individuals are more evenly distributed in the objective space. As Figure 18 illustrates, E^3-MOGA achieves slightly better CV than NSGA-II. X-E^3's CV is very high since X-E^3 is designed to find only extreme individuals.

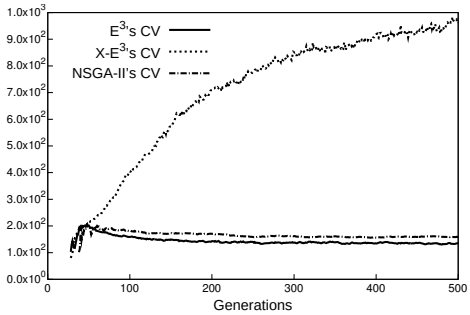

Fig. 18: Average CV of QoS Objective Values

Figure 19 illustrates the average ratio of individuals placed in the middle 50% range on each objective dimension. For example, when individuals yield the the maximum and minimum total costs of 900 and 100, respectively, the ratio is calculated based on the number of individuals that yield the total costs in between 300 and 800. When the ratio is closer to 50%,

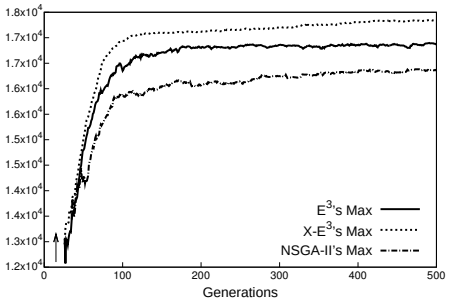

Fig. 6: Platinum Users' Throughput

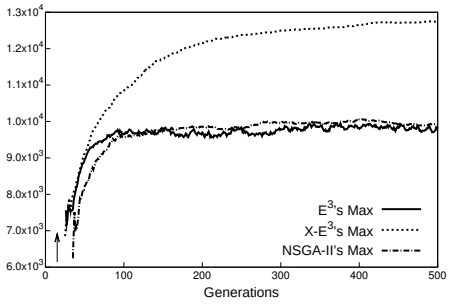

Fig. 7: Gold Users' Throughput

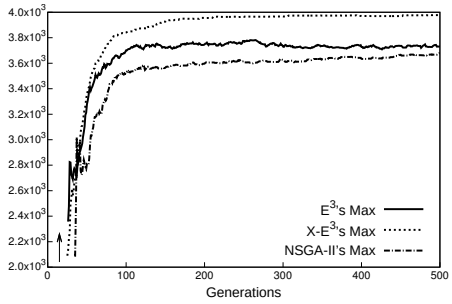

Fig. 8: Silver Users' Throughput

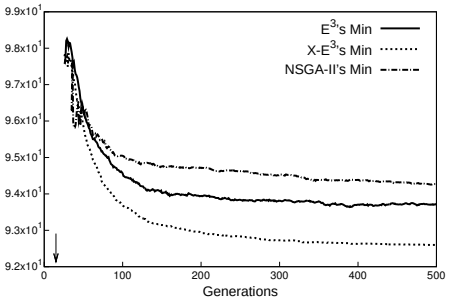

Fig. 9: Platinum Users' Latency

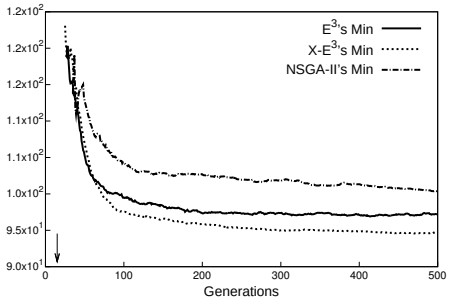

Fig. 10: Gold Users' Latency

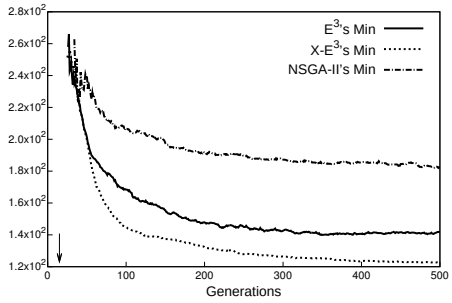

Fig. 11: Silver Users' Latency

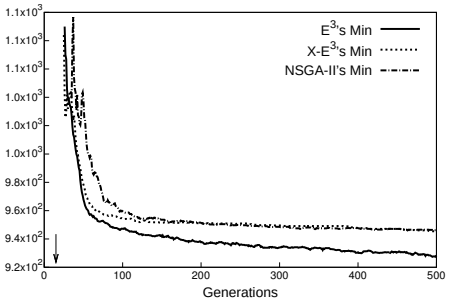

Fig. 12: Platinum Users' Cost

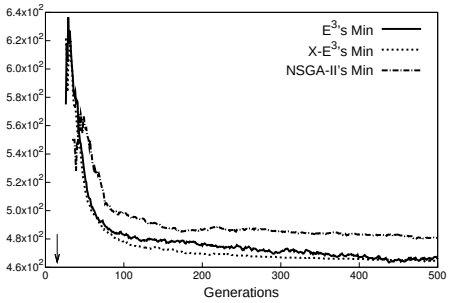

Fig. 13: Gold Users' Cost

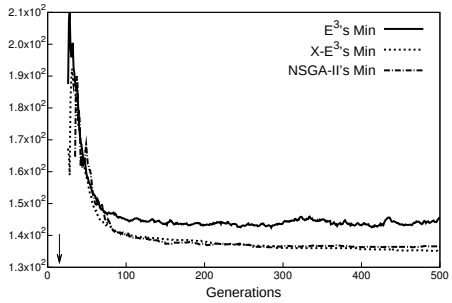

Fig. 14: Silver Users' Cost

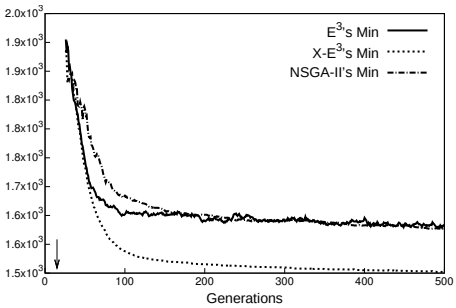

Fig. 15: Total Cost

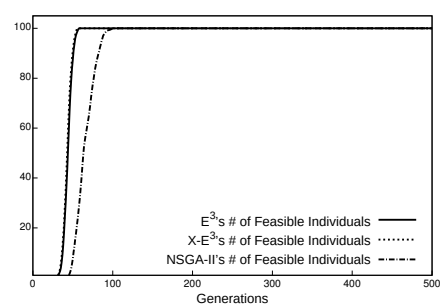

Fig. 16: # of Feasible Individuals

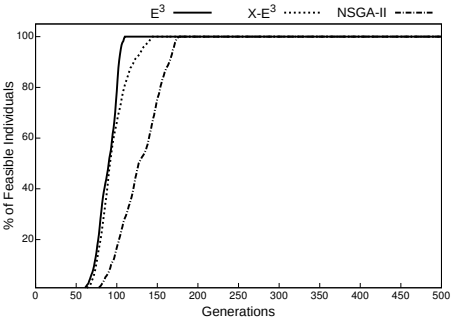

Fig. 17: # of Feasible Individuals with Stringent SLAs

individuals are more equally distributed in the middle and edge regions of the objective space. As Figure 19 illustrates, E^3-MOGA and NSGA-II distribute individuals equally in the middle and edge regions. (E^3-MOGA slightly outperforms NSGA-II.) In contrast, only 10% of X-E^3 individuals are placed in the middle 50% range on objective dimensions. This means that most individuals are placed in the edge (i.e., extreme) regions of the objective space as X-E^3 is designed to do so.

Figure 20 illustrates the variance of the number of individuals in an adaptive grid in the objective space. An adaptive grid is constructed by finding the maximum and minimum values in each objective dimension, dividing each dimension equally into 10 between the maximum and minimum, and constructing 10 dimensional hypercubes of 10^{10}. Figure 20 shows the variance of the number of individuals located in those hypercubes. E^3-MOGA yields a significantly lower variance than NSGA-II. In fact, it achieves the variance of zero over generations. This means that E^3-MOGA individuals are distributed much more evenly in the objective space than

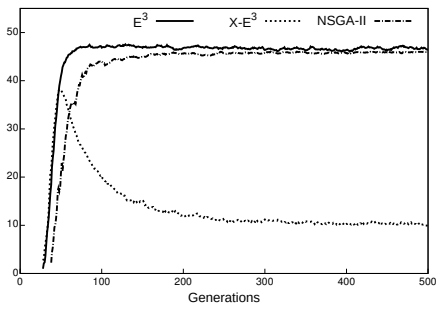

Fig. 19: # of Individuals in the Middle 50% Range on Obective Dimensions

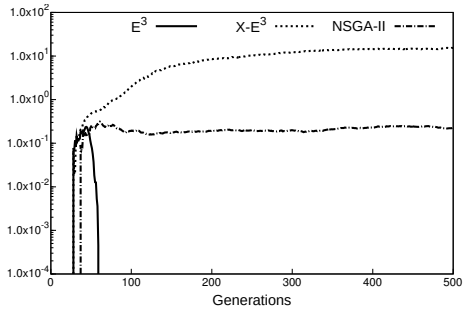

Fig. 20: Variance of # of Individuals in an Adaptive Grid

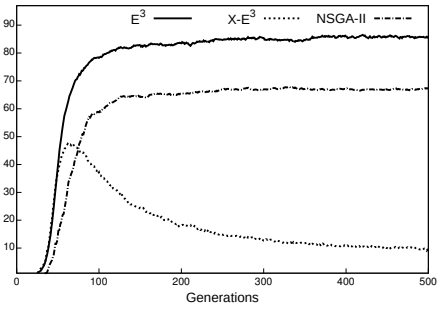

Fig. 21: Coverage in a Fixed Grid

NSGA-II individuals. X-E^3 yields a higher variance than E^3-MOGA and NSGA-II, which means X-E^3 individuals are clustered (around edge regions) in the objective space.

In order to measure the extent that individuals cover in the objective space, Figure 21 shows the number of hypercubes that contain individuals in a fixed grid in the objective space. A fixed grid is constructed by dividing each objective dimension by a certain length. E^3-MOGA individuals exist in 85 hypercubes at the 500th generation, while NSGA-II individuals exist in 65 hypercubes. This result indicates that E^3-MOGA covers a larger area in the objective space than NSGA-II.

Table 7 compares the hypervolumes that the individuals of E^3-MOGA, X-E^3 and NSGA-II exclusively dominate in the objective space. $v(A,B) = v_A/v_{AB}$ where v_A denotes the hypervolume that the individual set A dominates but the individual set B does not in the objective space. v_{AB} denotes the hypervolume that the individual sets A and B dominate in the objective space. 100 - $v(A,B)$ - $v(B,A)$ indicates the fraction of the hypervolume that both individual sets A and B dominate in v_{AB}. QoS objective values are normalized when they have different scales.

v(E^3-MOGA, NSGA-II) is 78.9% while v(NSGA-II, E^3-MOGA) is 12.1%. This means that E^3-MOGA explores the objective space better than NSGA-II and therefore exclusively dominates a larger portion in the objective space than NSGA-II. v(X-E^3, NSGA-II) is 2.3% while v(NSGA-II, X-E^3) is 97.5%. This is because X-E^3 is not designed to spread individuals widely in the objective space but to seek extreme solutions.

TABLE 7: Exclusively Dominated Hypervolume $v(A,B)$

A \ B	E^3-MOGA	X-E^3	NSGA-II
E^3-MOGA	-	-	78.9
X-E^3	-	-	2.3
NSGA-II	12.1	97.5	-

Figures 18 to 21 and Table 7 demonstrate that E^3-MOGA and X-E^3 perform as they are designed and E^3-MOGA explore the objective space better than NSGA-II by distributing individuals more equally and widely in the objective space.

Figure 22 shows three example solutions that E^3-MOGA produces and their QoS measures. As shown in the figure, the service compositions for platinum users tend to use high-end concrete services that yield higher performance with higher expenditure. In contrast, the service compositions for silver users tend to use low-end concrete services that yield lower performance with lower expenditure. The three individuals in Figure 22 also exhibits diverse trade-offs. The first individual intends to optimize platinum users' throughput, and the second one intends to optimize the total cost. If an application developer would like to optimize QoS measures under his/her budget constraint, he/she may choose the first one. He/she may choose the second one if he/she would like to save expenditure as long as given SLAs are satisfied. The third individual intends to optimize all QoS measures in a well-balanced manner.

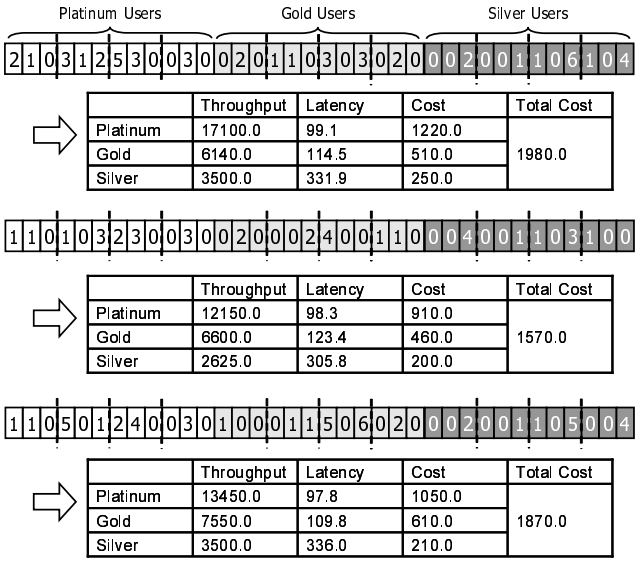

Fig. 22: Example Individuals that E^3-MOGA produces

5.2 Search Space and Complexity Analysis

The second simulation study evaluates how the SSC problem's complexity impacts the performance of E^3-MOGA, X-E^3 and NSGA-II. This simulation study simulates workflows, each of which contains a varying number of sequential abstract services. Each abstract service has three corresponding concrete services, and each concrete service can be deployed with at most 10 instances. Therefore, the size of search space (i.e., the total number of possible combinations of service instances) is $(10^3 - 1)^M$ for a single user category where M denotes the

number of abstract services in a workflow. The search space is expanded to $((10^3-1)^M)^3 \cong 1.0\times10^{9M}$, assuming three user categories. The search space of a workflow shown in FIgure 5 is approximately 1.0×10^{36}. As these numbers show, the size of search space tends to be huge in the SSC problem even if a workshop is structured with a small number of abstract and concrete services.

Table 8 shows a set of available concrete services for each abstract service. This simulation study assumes that the QoS values of each concrete service are constant. (QoS probability distributions are not assumed as in Table 4.) Table 9 shows SLAs for three user categories. (M is the number of abstract services in a workflow.) All algorithms are configured to run 1,000 generations. Other simulation configurations are same as the ones in the previous simulation study.

TABLE 8: QoS of Concrete Services

Concrete Service	QoS measures		
	Throughput	Latency	Cost
1	10,000	60	100
2	5,500	100	50
3	2,000	200	20

TABLE 9: SLAs (QoS Constraints)

User Category	SLAs			
	Throughput (Lower Bound)	Latency (Upper Bound)	Cost (Upper Bound)	Total Cost (Upper Bound)
Platinum	40,000	$80 \times M$	N/A	$1{,}000 \times M$
Gold	20,000	$120 \times M$	N/A	
Silver	15,000	N/A	$200 \times M$	

Figure 23 illustrates the number of generations that each algorithm requires to evolve all individuals to be feasible. Data points in the figure show the results with 1, 3, 5, ..., 25 abstract services in a workflow. As the complexity of the SSC problem grows, each of E^3-MOGA, X-E^3 and NSGA-II requires a linearly increasing number of generations to evolve all individuals to be feasible. E^3-MOGA and X-E^3 evolve individuals significantly faster than NSGA-II when the size of search space is larger. Consistent with the results in Figures 16 and 17 Figure 23 demonstrates that E^3-MOGA and X-E^3 are scalable to the size of search space.

In all 100 simulation runs, NSGA-II fails to evolve all individuals to be feasible in 1,000 generations when a workflow consists of 25 abstract services. It also fails in 20 and 60 simulation runs when a workflow consists of 21 and 23 abstract services, respectively. This is because NSGA-II does not consider SLA violations when it evaluates infeasible solutions. As Figure 23 shows, E^3-MOGA and X-E^3 handle infeasible individuals better and thereby evolve infeasible individuals to be feasible faster.

Figure 24 shows the time for E^3-MOGA, X-E^3 and NSGA-II to evolve all individuals to be feasible. E^3-MOGA and X-E^3 evolve individuals faster than NSGA-II when the size of search space is larger. It takes less than 20 seconds for E^3-MOGA and X-E^3 to evolve all individuals to be feasible in a workflow with 25 abstract services, while It takes approximately 105 seconds for NSGA-II in a workflow with 23 abstract services. Similar to the results in Figure 23, Figure 24 demonstrates that E^3-MOGA and X-E^3 are scalable to the size of search space.

Figure 24 also shows the time for the Simplex linear programming algorithm[3] to find an optimal solution. In this simulation study, Simplex is configured to seek a solution that maximizes the throughput for platinum users because linear programming is limited to seek a single optimal solution with respect to a single optimization objective (rather than Pareto-optimal solutions). Therefore, Simplex solves significantly simpler SSC problems than E^3-MOGA, X-E^3 and NSGA-II. As Figure 24 illustrates, Simplex operates efficiently when the size of search space is small; however, its computational time significantly increases as the size of search space grows. When a workflow consists of 16 abstract services, Simplex requires over 12 hours to find only one optimal solution. Given this observation, E^3-MOGA and X-E^3 are efficient to seek Pareto-optimal solutions in a reasonably short amount of time.

Figures 25, 26, 28 and 28 show the average CV of QoS values, the average ratio of individuals placed in the middle 50% range on each objective dimension, the variance of the number of individuals in an adaptive grid on the objective space, and the number of hypercubes that contain individuals in a fixed grid on the objective space, respectively. All results are obtained at the 1,000th generation. As these figures illustrate, the complexity of the SSC problem has little impacts on the quality of solutions that E^3-MOGA and NSGA-II produce while it has large impacts on X-E^3. X-E^3 tends to search very limited area of the search space since it does not consider the distribution of solutions. When the search space is small, X-E^3 tends to cluster individuals in very limited areas in the objective space. This results in a higher CV in Figure 25, a higher variance in Figure 27 and a lower coverage in Figure 28.

5.3 Optimality Analysis

E^3-MOGA, X-E^3 and NSGA-II are heuristic optimization algorithms, which are designed to seek Pareto-optimal solutions from a huge search space in a reasonably short time. Therefore, the individuals they produce may not be truly optimal. The third simulation study evaluates the optimality of individuals by comparing them with the truly optimal solutions. This simulation study simulates a workflow that has one abstract service associated with three concrete services. Each concrete service can be deployed with up to 10 instances. The size of search space is $(10^3-1)^3 \cong 1.0\times10^9$. All algorithms are configured to run 200 generations. Other simulation configurations are same as the ones in the previous simulation study.

Table 10 shows the maximum and minimum QoS values of the truly optimal solutions that an exhaustive (brute force) search algorithm finds. Tables 11 and 12 show the maximum and minimum QoS measures that E^3-MOGA and X-E^3 yield at the 200th generation. During 200 generations, E^3-MOGA and X-E^3 examine 20,100 individuals in total. Compared with the size of search space (1.0×10^9), E^3-MOGA and X-E^3 search very limited (only 0.002%) regions in the search space. This means that E^3-MOGA and X-E^3 are dramatically

[3]This paper uses GNU Linear Programming Kit (www.gnu.org/software/glpk/)

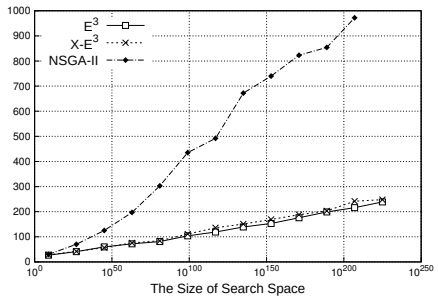

Fig. 23: # of Generations required to evolve all Individuals to be Feasible

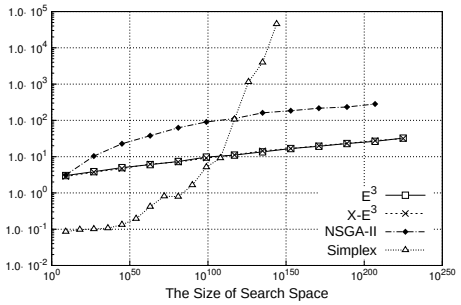

Fig. 24: Processing Time (seconds)

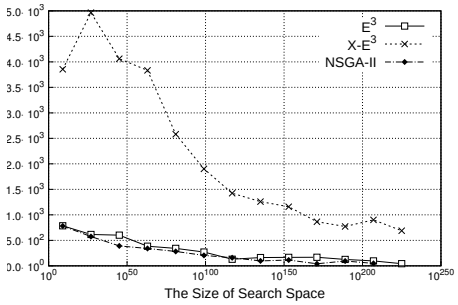

Fig. 25: Average CV of QoS Objective Values

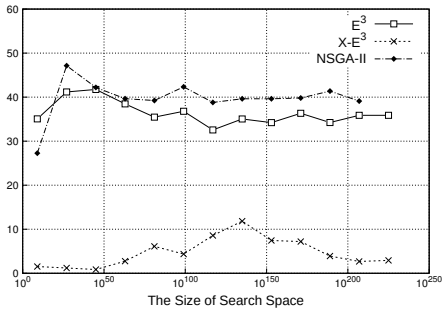

Fig. 26: # of Individuals in the Middle 50% Range on Objective Dimensions

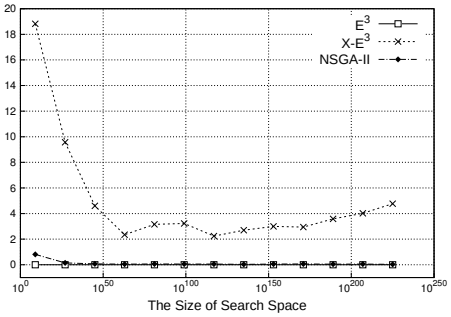

Fig. 27: Variance of # of Individuals in an Adaptive Grid

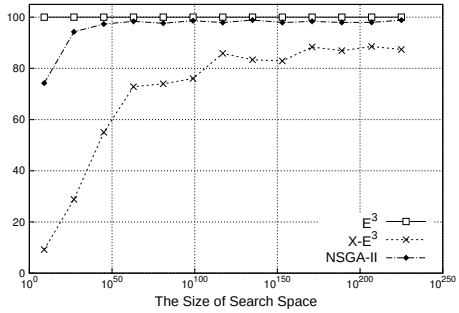

Fig. 28: Coverage in a Fixed Grid

TABLE 10: The Optimal QoS Values

User category	QoS Attribute	Maximum	Minimum
Platinum	Throughput	36000.0	24000.0
	Latency	80.0	60.0
	Cost	450.0	280.0
Gold	Throughput	33000.0	13000.0
	Latency	120.0	60.0
	Cost	320.0	150.0
Silver	Throughput	16500.0	8000.0
	Latency	200.0	80.0
	Cost	150.0	100.0
	Total Cost	700.0	530.0

TABLE 11: QoS Measures that E^3-MOGA yields

User	QoS Attr.	Max	Min	Error (%)	Coverage (%)
Platinum	Throughput	35437.5	24000.0	4.7	95.3
	Latency	80.0	61.0	5.0	95.0
	Cost	418.8	280.0	0.0	81.6
Gold	Throughput	31125.0	13000.0	9.4	90.6
	Latency	120.0	60.0	0.0	100.0
	Cost	311.3	150.0	0.0	94.7
Silver	Throughput	16500.0	8000.0	0.00	100.0
	Latency	200.0	111.3	26.0	74.0
	Cost	150.0	100.0	0.0	100.0
	Total Cost	700.0	538.8	5.2	94.9
Average				5.0	92.6

TABLE 12: QoS Measures that X-E^3 yields

User	QoS Attr.	Max	Min	Error (%)	Coverage (%)
Platinum	Throughput	36000.0	24000.0	0.00	100.00
	Latency	80.0	60.0	0.00	100.00
	Cost	420.0	280.0	0.00	82.35
Gold	Throughput	32687.5	13000.0	1.56	98.44
	Latency	108.75	60.0	0.00	81.25
	Cost	300.0	150.0	0.00	88.24
Silver	Throughput	16500.0	11000.0	0.00	64.71
	Latency	100.0	87.5	6.25	10.42
	Cost	150.0	100.0	0.00	100.00
	Total Cost	700.0	530.00	0.00	100.00
Average				0.78	82.54

faster in their optimization speed than an exhaustive search algorithm. Their search time is approximately 3 minutes while an exhaustive search algorithm's is over 12 hours.

Tables 11 and 12 also show the errors and coverage on each objective dimension. An error value indicates the distance between the optimal QoS value and the best QoS measure that E^3-MOGA or X-E^3 yields. For example, since throughput is an objective to be maximized, a throughput error indicates the distance between the maximum throughput values that the optimal solutions and E^3-MOGA/X-E^3 individuals yield. A smaller error indicates that the individuals of E^3-MOGA/X-E^3 are closer to the optimal solutions. A coverage value indicates how much E^3-MOGA/X-E^3 individuals cover the QoS value range that the optimal solutions yield (i.e., the range between the maximum and minimum QoS values that the optimal solutions yield). For example, the optimal solutions yield the maximum and minimum throughput of 36000.0 and 24000.0, respectively, for platinum users (Table 10). E^3-MOGA yields 35437.5 and 24000.0 (Table 11). Thus, the coverage of E^3-MOGA individuals is 95.3% by comparing the two max-min throughput range. A larger coverage indicates that the diversity of E^3-MOGA/X-E^3 individuals is similar to that of the optimal solutions.

Table 11 demonstrates that E^3-MOGA yields the average error of 5% and the average coverage of 92.6%. Table 12 illustrates that X-E^3 yields the average error of 0.78% and the average coverage of 82.5%. E^3-MOGA and X-E^3 produce quality individuals, which are close to the optimal solutions in both optimality and diversity, by searching only 0.002% of the search space. Compared with E^3-MOGA, X-E^3's error is better and coverage is worse. This is because X-E^3 is designed

to seek extreme individuals rather than balanced individuals. Table 13 shows that NSGA-II yields the average error of 10.87% and the average coverage of 84.67%. E^3-MOGA and X-E^3 outperform NSGA-II in optimality and diversity.

TABLE 13: QoS Measures that NSGA-II yields

User	QoS Attr.	Max	Min	Error (%)	Coverage (%)
Platinum	Throughput	32812.5	24000.0	26.66	73.44
	Latency	80.0	64.25	21.25	78.75
	Cost	398.75	280.0	0.00	69.85
Gold	Throughput	29937.5	13375.0	15.31	82.81
	Latency	120.0	65.0	8.33	91.67
	Cost	306.25	150.0	0.00	91.91
Silver	Throughput	15625.0	8187.5	10.29	87.50
	Latency	196.88	98.75	15.63	81.77
	Cost	150.0	101.25	2.50	97.50
	Total Cost	700.0	545.00	8.82	91.18
Average				10.87	84.64

5.4 Combination of E^3-MOGA and X-E^3

In order to evaluate how E^3-MOGA and X-E^3 complement each other, the forth simulation study combines the individuals that E^3-MOGA and X-E^3 produce and compares them with the individuals that NSGA-II produces. This study runs E^3-MOGA and X-E^3 independently for 250 generations each while NSGA-II for 500 generations. It uses the workflow and concrete services that the first simulation study uses (Figure 5 and Table 4). Other simulation configurations are same as the ones used in the previous simulation studies.

Table 14 shows the maximum and minimum QoS measures that a combination of E^3-MOGA and X-E^3 (E^3-MOGA + X-E^3) and NSGA-II yield. E^3-MOGA + X-E^3 exhibits 9% to 67% wider coverage (i.e., range between the maximum and minimum QoS values) than NSGA-II on QoS objective dimensions. On average, the coverage of E^3-MOGA + X-E^3 individuals is 33.8% higher than NSGA-II individuals. This is mainly because X-E^3 finds extreme solutions than NSGA-II even if it runs only for 250 generations.

TABLE 14: QoS Measures and Coverage

User category	QoS Attribute	NSGA-II Max	NSGA-II Min	E^3-MOGA + X-E^3 Max	E^3-MOGA + X-E^3 Min	Coverage (%)
Platinum	Thr.	16178.4	12230.5	17286.0	12150.0	130.1
	Latency	99.4	94.5	99.8	92.9	139.2
	Cost	1246.4	950.3	1331.5	928.3	136.2
Gold	Thr.	9943.2	6258.7	12401.1	6176.0	167.0
	Latency	128.5	100.6	129.3	94.8	123.5
	Cost	827.6	484.7	886.4	455.6	125.7
Silver	Thr.	3740.1	2204.2	4043.3	2113.6	125.6
	Latency	292.1	185.8	294.1	124.0	159.9
	Cost	247.7	141.1	250.0	133.8	109.0
	Total Cost	1997.5	1628.2	2000.0	1558.4	119.6
Average						133.8

Table 15 compares the hypervolumes that E^3-MOGA + X-E^3 and NSGA-II dominate exclusively. See Section 5.1 for more about $v(A,B)$. v(E^3-MOGA + X-E^3, NSGA-II) is approximately three times larger than v(NSGA-II, E^3-MOGA + X-E^3). This is mainly because E^3-MOGA seeks a larger region than than NSGA-II in the objective space even if it runs only for 250 generations.

Tables 14 and 15 clearly illustrate that E^3-MOGA and X-E^3 complement each other well. A combination of the two algorithms can produce higher quality solutions than NSGA-II in QoS measures, QoS coverage and the distribution of individuals.

TABLE 15: Exclusively Dominated Hypervolume $v(A,B)$

A \ B	E^3-MOGA + X-E^3	NSGA-II
E^3-MOGA + X-E^3	-	67.3
NSGA-II	22.5	-

6 Related Work

This paper describes a set of extensions to the authors' prior work [22]. This paper revises the fitness function of E^3-MOGA so that it can represent the quality of solutions more properly. Also, this paper newly proposes X-E^3 to explore extreme solutions, which the previous work [22] did not consider. Moreover, this paper carries out more extensive simulation studies than the previous work [22] in order to reveal the performance characteristics of E^3-MOGA and X-E^3 in detail and their advantages over NSGA-II.

The SSC problem has been well studied. Blake et al. study an exhaustive search to solve the SSC problem [23]; however, this approach has a problem in its computational cost since the SSC problem tends to have a huge search space as shown in Section 5.

A large number of research work map the SSC problem to linear integer problems and solve them with linear programming algorithms [4]–[9]. This approach is not suitable enough to effectively solve the SSC problem for several reasons. First, linear programming intends to find only one optimal solution but not to reveal the trade-offs among conflicting QoS objectives. It also suffers from high computational costs especially when the size of search space is large. As Section 5.3 discusses, it is less scalable compared to genetic algorithms [3]. Wiesemann et al. map the SSC problem to a set of mixed linear integer problems [25] to examine the trade-offs among QoS objectives. However, it does not address the scalability issue.

Linear programming is designed to find a single optimal solution that optimizes a fitness function defined in a linear form. Most linear programming algorithms use the Simple Additive Weighting (SAW) method to define their fitness functions. In order to solve the SSC problem, it is not always straightforward to determine weight values in a fitness function because objectives often conflict with each other and have different value ranges. In fact, it is known that the SAW method does not work well for non-trivial optimization problems [24].

Another limitation of linear programming to solve the SSC problem is that it is hard to define certain QoS attributes (e.g., latency of redundant parallel service instances) in a linear form. Guo et al. map the SSC problem to a mixed nonlinear integer problem to define QoS attributes in a nonlinear form [26]; however, they consider only one objective and seek a single optimal solution rather than Pareto-optimal solutions.

In contrast to existing linear programming approaches, E^3-MOGA and X-E^3 are designed to seek Pareto-optimal solutions with respect to conflicting QoS objectives and explore

the trade-offs among them. They are more scalable to the size of search space than linear programming (Section 5.3). E^3-MOGA and X-E^3 do not use the SAW method to compute fitness values for individuals. Instead, they employ the notion of domination ranking. Moreover, they have no limitations to define QoS attributes in both linear and nonlinear forms.

A number of research work have proposed heuristics to solve the SSC problem and other problems in service-oriented applications [4], [5], [26], [27]. For example, Menascé et al. propose a heuristic algorithm to solve the SSC problem with respect to execution time and execution cost of an application [27]. They use the SAW method to define a fitness function and do not aim to seek Pareto-optimal solutions.

Nguyen et al. [28] and Lin et al. [29] avoid the SAW method. They leverage fuzzy logic to eliminate weight values in fitness functions. However, their algorithms are designed to seek a single optimal solution rather than multiple Pareto-optimal solutions.

Various research efforts have exploited classical genetic algorithms (i.e., genetic algorithms that do not use domination ranking) to solve the SSC problem [3], [14]–[16]. Liu et al. leverage a single-objective particle swarm optimization algorithm [30]. These existing algorithms use the SAW method to define their fitness functions and do not intend to seek multiple Pareto-optimal solutions.

A few attempts exist to solve the SSC problem with a few multiobjective genetic algorithms [10]–[13]. They assume simpler service composition models than E^3-MOGA and X-E^3. For example, SLAs (i.e., QoS constraints) are not considered in [10]–[12]. QoS probability distributions are not considered in [10], [12], [13]. In contrast, E^3-MOGA and X-E^3 uses a more realistic service composition model that reflects a set of assumptions in service-oriented applications especially in cloud computing environments.

Claro et al. uses NSGA-II [10], which E^3-MOGA and X-E^3 outperform. Taboada et al. [11] and Liu et al. [13] do not use elitism in selection of individuals, while E^3-MOGA and X-E^3 do because it is known that elitism improves convergence speed in genetic algorithms [24], [31]. E^3-MOGA normalizes objective values and calculates the Manhattan distance between individuals in order to distribute them in the objective space in a balanced way. No existing multiobjective genetic algorithms attempt to do this for the SSC problem. Chang et al. seek non-dominated solutions that minimize the distance to the Utopian point (i.e., a predefined desirable point in the objective space) [12]. This favors individuals that yield intermediate objective values and misses the individuals that yield extreme objective values. In contrast, E^3-MOGA seeks well-distributed individuals, and X-E^3 seeks extreme individuals. Moreover, no existing multiobjective genetic algorithms for the SSC problem have been extensively evaluated with, for example, search space analysis, optimality analysis and comparison with other algorithms, as this paper does.

7 Conclusion

This paper proposes an optimization framework, called E^3, to solve the SSC problem. E^3 defines a service composition model and provides multiobjective genetic algorithms: E^3-MOGA and X-E^3. The service composition model in E^3 is practical and well-applicable to any computing environments that differentiate resource provision for services; for example, Amazon EC2, GoGrid[4] and Microsoft Azure[5]. Simulation evaluation results demonstrate that E^3 finds quality solutions in a reasonably short time and E^3 consistently outperforms NSGA-II, which is a well-known multiobjective genetic algorithm.

Several extensions are planned as future work. The current E^3 aggregates QoS values in a deterministic manner. It is known that a deterministic scheme can lead to too pessimistic aggregated QoS values [32] and preclude the use of probabilistic SLAs[6]. To address this issue, E^3 will be extended to aggregate QoS values using Monte Carlo methods and support probabilistic SLAs.

Also, E^3 will be extended to support the notion of computing resources such as hosts. Given this notion, E^3 will be enhanced to optimize service compositions that consider how many hosts to purchase and which services instances to deploy on which hosts.

8 Acknowledgement

This work is supported in part by OGIS International, Inc.

References

[1] M. Bichler and K. Lin, "Service-Oriented Computing," *IEEE Computer*, vol. 39, no. 6, June 2006.

[2] M. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," in *IEEE International Conference on Web Information Systems Engineering*, December 2003.

[3] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "An Approach for QoS-aware Service Composition based on Genetic Algorithms," in *ACM International Conference on Genetic and Evolutionary Computation*, Ontario, Canada, June 2005, pp. 1069–1075.

[4] J. Anselmi, D. Ardagna, and P. Cremonesi, "A QoS-based Selection Approach of Autonomic Grid Services," in *ACM Workshop on Service-Oriented Computing Performance*, Monterey, CA, June 2007, pp. 1–8.

[5] T. Yu, Y. Zhang, and K. J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on The Web*, vol. 1, no. 1, pp. 129–136, December 2007.

[6] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, June 2007.

[7] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti, "Flow-based Service Selection for Web Service Composition Supporting Multiple QoS Classes," in *IEEE International Conference on Web Services*, Salt Lake City, UT, July 2007, pp. 743–750.

[8] Y. Qu, C. Lin, Y. Z. Wang, and Z. Shan, "QoS-Aware Composite Service Selection in Grids," in *International Conference on Grid and Cooperative Computing*, Hunan, China, October 2006, pp. 458–465.

[9] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 20, no. 5, pp. 311–327, May 2004.

[10] D. B. Claro, P. Albers, and J. Hao, "Selecting Web Services for Optimal Composition," in *IEEE International Workshop on Semantic and Dynamic Web Processes*, Orland, FL, July 2005, pp. 32–45.

[4] www.gogrid.com

[5] www.microsoft.com/windowsazure/

[6] A probabilistic SLA defines a set of QoS constraints probabilistically. For example, it may declare that throughput must be above a certain value in 90% of the cases.

[11] 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*, vol. 57, no. 1, pp. 182–191, March 2008.

[12] W. Chang, C. Wu, and C. Chang, "Optimizing Dynamic Web Service Component Composition by Using Evolutionary Algorithms," in *IEEE/ACM International Conference on Web Intelligence*, Septemer 2005.

[13] S. Liu, Y. Liu, N. Jing, G. Tang, and Y. Tang, "A Dynamic Web Service Selection Strategy with QoS Global Optimization based on Multi-Objective Genetic Algorithm," in *International Conference on Grid and Cooperative Computing*, November 2005.

[14] M. C. Jaeger and G. Mühl, "QoS-based Selection of Services: The Implementation of a Genetic Algorithm," in *Workshop on Service-Oriented Architectures and Service-Oriented Computing*, Bern, Switzerland, March 2007, pp. 359–370.

[15] C. Gao, M. Cai, and H. Chen, "QoS-Aware Service Composition based on Tree-Coded Genetic Algorithm," in *IEEE International Computer Software and Applications Conference*, July 2007.

[16] Y. Gao, B. Zhang, J. Na, L. Yang, Y. Dai, and Q. Gong, "Optimal Selection of Web Services with End-to-End Constraints," in *IEEE International Conference on Grid and Cooperative Computing*, October 2006.

[17] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Professional, 1989.

[19] N. Srinivas and K. Deb, "Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[21] H. Ishibuchi, N. Tsukamoto, Y. Hitotsuyanagi, and Y. Nojima, "Effectiveness of Scalability Improvement Attempts on the Performance of NSGA-II for Many-Objective Problems," in *ACM International Conference on Genetic and Evolutionary Computation*, Atlanta, GA, July 2008, pp. 649–656.

[22] H. Wada, P. Champrasert, J. Suzuki, and K. Oba, "Multiobjective Optimization of SLA-aware Service Composition," in *IEEE Workshop on Methodologies for Non-functional Properties in Services Computing*, July 2008.

[23] M. Blake and D. Cummings, "Workflow Composition of Service Level Agreements," in *IEEE International Conference Services Computing*, July 2007.

[24] C. A. C. Coello, "Recent Trends in Evolutionary Multiobjective Optimization," in *Evolutionary Multiobjective Optimization*, L. Jain, X. Wu, A. Abraham, L. Jain, and R. Goldberg, Eds. Springer, 2005, pp. 7–32.

[25] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A Stochastic Programming Approach for QoS-Aware Service Composition," in *IEEE International Symposium on Cluster Computing and The Grid*, May 2008.

[26] H. Guo, J. Huai, H. Li, T. Deng, Y. Li, and Z. Du, "ANGEL: Optimal Configuration for High Available Service Composition," in *IEEE International Conference on Web Services*, July 2007.

[27] D. Menascé, E. Casalicchio, and V. Dubey, "A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures," in *ACM International Workshop on Software and Performance*, June 2008.

[28] X. Nguyen, R. Kowalczyk, and M. Phan, "Modelling and Solving QoS Composition Problem Using Fuzzy DisCSP," in *IEEE International Conference on Web Services*, Septemer 2006.

[29] M. Lin, J. Xie, H. Guo, and H. Wang, "Solving Qos-Driven Web Service Dynamic Composition as Fuzzy Constraint Satisfaction," in *IEEE International Conference on e-Technology, e-Commerce and e-Service*, March 2005.

[30] J. Liu, J. Li, K. Liu, and W. Wei, "A Hybrid Genetic and Particle Swarm Algorithm for Service Composition," in *IEEE International Conference on Advanced Language Processing and Web Information Technology*, Septemer 2007.

[31] A. Konaka, D. W. Coitb, and A. E. Smithc, "Multi-Objective optimization Using genetic algorithms: A tutorial ," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, Septemer 2006.

[32] S. Rosario, A. Benveniste, S. Haar, and C. Jard, "Probabilistic QoS and Soft Contracts for Transaction-based Web Services Orchestrations," *IEEE Transactions on Services Computing*, vol. 1, no. 4, pp. 187–200, 2008.