Single-Objective versus Multi-Objective Genetic Algorithms for Workflow Composition based on Service Level Agreements

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
simone.ludwig@ndsu.edu

Abstract—**Workflow composition is a very important issue for service-oriented environments. In particular, the composition of services based on quality of service (QoS) attributes has been gaining attention. The user can describe the request of a workflow in terms of QoS attributes, i.e., the user aims for specific service performance, e.g. low waiting time, high reliability and availability, which are based on different service level plans provided by the providers. Past research has addressed this workflow composition problem classifying them into single-objective and multi-objective optimization solutions. Most of the research has employed the single-objective approach whereby the different objectives, i.e., QoS attributes, are aggregated by a weighted approach. Fewer research approaches have used the multi-objective approach, whereby several solutions are produced by a set of Pareto solutions that have equivalent quality to satisfy specific service level agreements. However, no comparison has been done investigating both approaches. Therefore, this paper addresses this shortcoming by an analysis of both, measuring the success ratio as well as the execution time of two Genetic algorithm implementations (single-objective and multi-objective) applied to the workflow composition problem.**

Genetic algorithm; quality of service; service level agreements; workflow composition; pareto-efficiency

I. INTRODUCTION

Even though the web was initially intended for human use, however, it can be said, that the web has evolved over the years, in particular with the introduction of web services. Web services introduced a higher-level functionality to make the web dynamic, as well as it enabled configurable software applications to improve productivity. Service-based applications consist of three components, which are provider, consumer and registry. Providers publish their services in registries, whereas consumers invoke the services after looking them up in the registry [1].

A service-oriented environment has special characteristics that distinguishes it from other computing environments: (i) the environment is dynamic - indicating that service providers are non-persistent and may become unavailable unpredictably. This means the environment will change over time as the system operates. The same principle is applied for service consumers; (ii) the number of service providers is unbounded; (iii) services are owned by various stakeholders with different aims and objectives. There may be unreliable, insecure or even malicious service providers; (iv) there is no central authority that can control all the service providers and consumers; (v) service providers and consumers are self-interested. In a service rich environment, it is necessary to provide support for automated service discovery. This is necessary to enable direct interaction between software sub-systems (acting as consumers and providers).

Due to the changing nature of service-oriented environments, the ability to locate services of interest in such an open, dynamic, and distributed environment has become an essential requirement. Traditional approaches to service discovery and selection have generally relied on the existence of pre-defined registry services, which contain descriptions that follow some shared data model. Often the description of a service is also very limited in such registry services, with little or no support for problem-specific annotations that describe properties of a service.

Current service-oriented architecture standards mainly rely on functional properties, however, the service registries lack mechanisms for managing services' non-functional properties. Such non-functional properties are expressed in terms of QoS parameters. QoS for web services allows consumers to have confidence in the use of services by aiming to experience good service performance, such as waiting time, reliability, and availability. It is difficult for service consumers to choose services from service registries, which contain hundreds of similar web services, given that the selection is only based on functional properties (even though they differ in the QoS values they deliver). In addition, QoS properties are dynamic in nature, and therefore, mechanisms are necessary for managing the dynamic changes of QoS properties [2].

A service level agreement (SLA) is a contract between a service provider and a service consumer, which captures the agreed-upon terms with respect to quality of service (QoS) parameters. Considering a service-oriented computing environment, capabilities are shared via the implementation of web services exposed by a service provider. When a service requester requires a specific functionality, which cannot be provided by one single service, the composition of multiple services needs to be done thereby creating a workflow. In addition, the composition of web services should not only be functionally compatible, but also should be compatible with regards to the defined service levels.

This paper addresses the comparison of a single-objective versus a multi-objective approach to the workflow composition problem. The single-objective Genetic algorithm (GA) approach uses a weighted method to combine the QoS parameters, and the multi-objective GA approach uses the idea of pareto-efficient solutions to find an appropriate selection of services for the workflows.

The remainder of this paper is structured as follows: Section 2 describes related work and motivates the work conducted in this paper. In Section 3, the two approaches being evaluated are described in detail. Section 4 presents the experiments and discusses the results. The concluding section summarizes the findings and gives an account to future work.

II. Related Work

Related work in the area of composition of workflows with particular focus on QoS is multifaceted. A framework for QoS-based web service contracting is proposed in [3]. The framework consists of an extensible model that defines domain-dependent and domain-independent QoS attributes, as well as a method to establish the contract phase is proposed. A matchmaking algorithm ranks the services that are functionally equivalent based on their ability to fulfill the requirements.

A dynamic web service selection and composition approach is described in [4]. The approach determines a subset of web services to be invoked during runtime in order to orchestrate a composite web service successfully. A finite state machine model is used to describe the permitted invocation sequences of the web service operations, and a reliability measure is aggregated for the web service operations.

The workflow service selection problem has received a lot of attention whereby many linear-programming approaches have been used [5,6]. In [7], complex workflow patterns are used to address the service selection problem. Linear programming is used to solve the optimization problem using an aggregation function for different QoS attributes.

In [8], a quality-driven web service composition approach is outlined applying linear programming. A global planning approach is employed to optimally select component services during the execution of a composite service. However, given that it is an exact method is does not scale very well for growing search spaces [9].

Genetic algorithm approaches have been applied to solve the QoS-aware service composition problem. Most of the approaches use the aggregated single-objective genetic algorithm approach by combining several objectives into a single objective value. There are different aggregation functions, sometimes also referred to as ranking function, used in literature that are linear normalization [10], vector normalization [10], and max-min [2] normalization. Linear normalization simply divides the values of each different QoS parameter by its maximum value in the case of maximization. For the case of minimization, the minimum is divided by the value of the QoS parameter. For vector normalization the values of the QoS parameters are divided by its norm. For minimization, one is divided by the QoS parameters as well as by its norm. The max-min normalization subtracts the minimum value of an attribute from each value of the QoS attributes and then divides the difference by the range of the attributes in the case of maximization; for minimization, the QoS value is subtracted by the maximum QoS value divided by the difference of the range of the QoS attributes. In this paper, we are using the max-min approach.

In [11], a genetic algorithm approach is used for four QoS parameters. Basic GA-parameters are varied such as mutation rate, number of generations, fitness function, penalty factor, as well as a comparison of the GA-approach to other heuristics is done. The study reveals that the GA offers a good overall performance, however, not as good when compared to the other heuristics such as branch-and-bound and exhaustive search. This is surprising as GA usually reaches very close to the optimal solution. Their work however does not address SLA.

In [9], another GA approach is discussed. They are using five QoS attributes and include a factor measuring the ratio of the generations, and the maximum generations as well in their fitness function. Similar to the approach above, no SLA is addressed. Their empirical study compares the GA approach with linear programming. They point out that the linear programming approach does not handle non-linear functions, which the GA method can. Another major point they make is that the GA scales well when the number of services increases.

There are also some multi-objective based approaches to tackle the workflow composition problem. In [12], a multi-objective non-dominated sorting GA (NSGA-II) is proposed. A set of Pareto optimal solutions is created that satisfy the user requirements. Their experimental results show that by using NSGA-II, several feasible solutions can be returned to the user allowing the user to select the appropriate solution.

[13] also implements a NSGA-II algorithm for the implementation of an optimal composition. Four QoS attributes are used of which reputation is tackled with a fuzzy logic based method. Again, no SLA is used.

In [14], a multi-objective approach to address the service composition problem with a SLA-aware approach is introduced. Three QoS parameters are considered and the fitness value of the multi-objective approach is based on domination rank and density. The domination rank increases the fitness value, whereas the density factor decreases the fitness value. The SLA-aware part of the approach on the concept of different service levels, i.e. Platinum, Gold and Silver is outlined. The simulation results present the different service levels outlining the maximum, minimum, average and lower bound values for all QoS parameters. However, there is no mention regarding execution time or scalability, and therefore, the suitability of the approach is not discussed.

No comparison between the single-objective and multi-objective workflow composition has been conducted. Also, not many of the provided research investigations have outlined the performance in terms of execution time, and addressed the scalability of the workflow composition problem. In addition, only one related evolutionary research approach used SLA. Therefore, a thorough analysis of a single-objective and multi-objective GA approach is conducted, as well as SLA is considered in this paper.

III. GA Approaches

A. *Workflow Example*

In order to show how abstract workflows are provided with the help of the workflow DB and the user's input request, the following sample workflow as shown in Figure 1 is provided, showing the abstract as well as the concrete services.

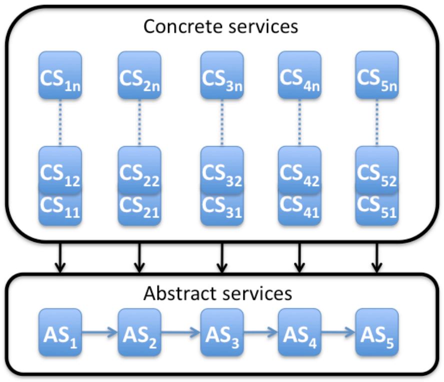

Figure 1. Example of an image processing workflow.

The figure displays an example of the processing of an image [19]. The process is as follows: the image is first read in (AS$_1$), and converted to Grey scale (AS$_2$), then it is thresholded (AS$_3$), i.e., image pixels less than a certain amount is set to black and everything higher is set to white. The difference is then taken between this image and an image where the white lines are thinned out, i.e., detail is taken away (AS$_4$). The resulting image is produced, by taking the difference between the detail of the white parts, which were pruned by the *ShrinkWhite* unit resulting in an outline of an image, and saved as a new compound component (AS$_5$).

The concrete services (CS$_{xy}$) for each abstract service (AS$_x$) provide the same functional, but different non-functional properties; i.e., QoS attributes. Selecting the services of a workflow based on QoS parameters requires an algorithm that can optimize the assignment of concrete services for a workflow for a given abstract workflow description. Furthermore, we are considering that multiple requests are being served at the same time. Given that performance in a service-oriented environment is of essence, we argue that we do not need to have optimal assignments, but close to optimal assignments should be found within a reasonable time.

B. *Quality of Service Metric and Objective Function*

There are many measures available for different QoS criteria, however, we consider the following four generic quality criteria for single services, also referred to as QoS parameters: reliability, availability, execution duration, and execution price. The first two QoS parameters are to be maximized, whereas the last two are to be minimized.

The *reliability* $q_1(s)$ of a service is the fraction of requests correctly responded to within a maximum expected time frame. Reliability is a measure related to the hardware and software configuration of Web services and their network connections. Reliability values are computed from past data measuring the successful executions in relation to the overall number of executions.

The *availability* $q_2(s)$ of a service is the fraction of time that the service is accessible. It measures the total amount of time in which the service is available during the last defined period of time (threshold is set by administrator).

The *execution duration* $q_3(s)$ denotes the expected delay in seconds from the moment a request is made until the moment when the results are returned. Services advertise their processing time or provide methods to inquire about it.

The *execution price* $q_4(s)$ represents the amount of money a user has to pay for executing a service. Web service providers usually advertise the execution price directly or they provide methods to inquire about it.

The QoS vector $q(s)$ of a service s is defined as follows:

$q(s) = (q_1(s), q_2(s), q_3(s), q_4(s))$.

However, in this study we are concerned not only with single services, but with complete workflows, and therefore the QoS parameters of the single services have to be aggregated. We assume in our study that we are only using sequential workflows. Therefore, the *availability* $Q_1(wf)$ and *reliability* $Q_2(wf)$ of a workflow wf is calculated as the product of each single service's availability and reliability respectively. The *execution duration* $Q_3(wf)$, and *execution price* $Q_4(wf)$ of a workflow wf is the average of each single service's execution duration and service cost respectively.

Therefore, the QoS vector $Q(wf)$ of a workflow wf is denoted as: $Q(wf) = (Q_1(wf), Q_2(wf), Q_3(wf), Q_4(wf))$.

Our goal is to maximize the selection of services within a workflow based on the QoS parameters. In addition, we are maximizing N workflows at the same time.

The single objective function for *reliability* and *availability* is:

$$f_{obj} = \frac{Q_{i,j} - Q_j^{\min}}{Q_j^{\max} - Q_j^{\min}}$$

and the single objective function for *execution duration* and *execution price* is:

$$f_{obj} = \frac{Q_j^{\max} - Q_{i,j}}{Q_j^{\max} - Q_j^{\min}}$$

whereby we define Q_{ij} to be the value for workflow i and the j^{th} QoS parameter, and we define $Q_j^{\max}$ to be the maximum score any of the considered services achieves for the j^{th} QoS parameter as defined above, i.e., $Q_j^{\max} = \max_{s \in S} q_j(s)$ where S is the set of all possible services. And similarly, $Q_j^{\min}$ to be the minimum score any of the considered services achieves for the j^{th} QoS parameter ($Q_j^{\min} = \min_{s \in S} q_j(s)$).

Given that the different service levels need to be taken into account we have to define the following constraints:

$Q_{ij} \geq Q_j(p) \quad for\ j = 1,2$

and

$Q_{ij} \leq Q_j(p) \quad for\ j = 3,4$

whereby $Q_j(p)$ is the j^{th} QoS value given the chosen service level plan p.

The overall objective function for the optimization of the workflows is the following:

$$f_{obj} = \max \sum_{i=1}^{N} \left(\sum_{j=1}^{2} w_j \left(\frac{Q_{i,j} - Q_j^{\min}}{Q_j^{\max} - Q_j^{\min}} \right) + \sum_{j=3}^{4} w_j \left(\frac{Q_j^{\max} - Q_{i,j}}{Q_j^{\max} - Q_j^{\min}} \right) \right)$$

Note that the individual QoS parameters are treated differently depending on whether its value is minimized or maximized. Normalized scores are used and each QoS parameter can be weighted differently by parameter w_j.

C. *Service Level Plan*

We define the SLA for each user category's reliability, availability, execution duration, and execution cost accordingly.

TABLE I. NORMALIZED SERVICE LEVELS FOR DIFFERENT SERVICE PLANS

	Reliability	**Availability**	**Time**	**Cost**
Platinum	0.6	0.5	0.7	0.5
Silver	0.7	0.7	0.5	0.65
Gold	0.8	0.9	0.3	0.8

Table 1 outlines the different service level plans (SLP) with normalized service level agreement values that are available to the users. Three different user categories are defined as Platinum, Silver and Gold.

D. *Single-objective GA Approach (GA)*

A GA is a heuristic used to find approximate solutions to difficult-to-solve problems by applying the principles of evolutionary biology to computer science problems. GAs use biologically-derived techniques such as inheritance, mutation, natural selection, and recombination (or crossover) [15]. GAs are typically implemented as a computer simulation in which a population of solutions (or individuals) to an optimization problem evolve towards better solutions. This is possible as each solution is a chromosome, which can undergo genetic modification.

In this study, each chromosome has the following structure:

SLP	CS1	CS2	CS3	SLP	CS1	CS2	CS3	CS4	…
2	13	26	31	1	18	23	37	42	…

A gene within the chromosome consists of integers, whereby the first number characterizes the SLP, and the following integers are the concrete services of the workflow. The first digit from the left of a concrete service (CS) characterizes the abstract service number; the second describes the specific concrete service implementation. The number of services the abstract workflow consists of determines the length of the gene. For example, the first workflow consists of 3 services, the second of 4 services, etc. The service level guides the fitness calculations as described in the previous subsection, i.e., the fitness value is determined depending on the service level. For this study, we consider workflows up to 5 services, having 10 concrete services available for each of the 10 abstract services.

The process of the optimization starts with a population of completely randomly generated individuals. In each generation, the fitness of each population member is evaluated. The fittest individuals, in terms of best fitness value, e.g. from an archive population, where the best solutions found so far are saved. As even the quality of solutions can range widely, particularly in earlier generations, members compete in tournaments, with winners forming a mating pool. Two parents are randomly selected from the pool, and undergo cycle crossover [16] and mutation to form two children. This is repeated until the new population of size N is filled. The new population is evaluated; its members compete for inclusion in the archive, and the process repeats until either a set number of generations are completed, stagnation, or termination criteria are met.

Configurable parameters in the implementation include number of iterations as termination criterion, tournament size (the size of the tournament used to select parents), crossover probability, effected positions (how many positions are set to crossover in the crossover mask), and mutation probability.

E. *Multi-objective GA Approach (NSGA-II)*

For the multi-objective approach NSGA-II was implemented. NSGA-II is a fast elitist non-dominated sorting genetic algorithm [17]. In NSGA-II [18] the most fit individuals from the union of archive and child populations are determined by a ranking mechanism (or crowded comparison operator) composed of two parts. The first part 'peels' away layers of non-dominated fronts, and ranks solutions in earlier fronts as better. The second part computes a dispersion measure, the crowding distance, to determine how close a solution's nearest neighbors are, with larger distances being better. It is employed here to search for better workflow compositions, which guide the evolutionary process toward solutions with better objective values.

Non-dominated solutions are desirable in the sense that it is impossible to find another solution in the set, which improves the value on any objective (i.e., QoS parameters) without simultaneously degrading the quality of the other objective, and is formally defined as follows:

Let $o_1, o_2, \ldots, o_n$ be the objective functions that are to be maximized. Let S be the set of obtained solutions. $s \in S$ is dominated by $t \in S$ (denoted $t \succ s$) if $\exists j,\ j \in \{1,\ldots,n\}$, such that $o_j(t) > o_j(s)$ and $\forall i$, $1 \leq i \leq n$, $o_j(t) \geq o_i(s)$. A non-dominated solution is therefore any solution $s \in S$, which is not dominated by any other $t \in S$.

The implemented NSGA-II algorithm differs from the previous GA as follows. After random initialization and the crossover/mutation phase, all individuals are ordered into non-dominated sets. First, a set of individuals that are not dominated is computed. Then, the non-dominated individuals are removed from the set of individuals for which non-domination is computed. After this, the reduced set is used to find the next non-dominated solutions, i.e., those solutions only dominated by the first set. The non-dominated sets get higher non-dominated set values. The search for non-dominated sets is called iteratively until enough individuals make up a generation and are sorted into non-dominated sets. Sets themselves are sorted using the crowding distance. Crowding distance is only computed within non-dominated sets. To compute the crowding distance, the following steps are followed for each objective:

- sort the individuals by the current objective.
- add the distance to the previous and next individual in the current objective to the crowding distance.

The sorting is implemented as a non-recursive quicksort algorithm. The last non-dominated set is truncated if the number of individuals in all calculated non-dominated sets is higher than the specified number of individuals. This sorting of non-dominated sets and truncation of the last set identifies NSGA-II's elitism. Tournament selection is used as the selection strategy. The individual with the best (smallest) non-dominated set wins the tournament. If two or more individuals share the lowest non-dominated value, the individual with the highest crowding distance from those individuals wins.

Configurable parameters in the implementation include maximum number of iterations, tournament size, crossover probability, elected positions, mutation probability, and lowest flag that defines whether the maximum crowding distance is also added if there is no following individual, i.e., end of the sorted list is reached for one objective.

IV. Experiments

A. Experimental Setup

Both approaches were implemented using Java. Experiments were designed to measure the success ratio and the execution time. The success ratio measures the percentage of successful workflow compositions in terms of fulfilled SLA. The algorithms were further analyzed with regard to the number of iterations, and the number of individuals used. All measurement points shown are average results taken from 30 runs to guarantee an equal distribution and statistical correctness. The data sets for the workflows and services were randomly generated, whereby workflows were created consisting of up to 5 abstract services, out of a pool of 10 concrete services for each of the 10 available abstract services (equals 100 concrete services). Please note that we assume that a particular concrete service can be used in several workflows. There is no maximum number given regarding how often one particular service can be called simultaneously.

The following parameter settings have been chosen due to their superior performance on the workflow composition problem. The GA/NSGA-II settings were: population size = 100, mutation probability = 0.05, crossover probability = 0.7, size of the tournament selection = 4, and number of positions that are selected for crossover = 0.1. The experiments were conducted on an Intel Core 2 Duo (2.4GHz, 3MB L2 cache) running the Java Version 1.6.0 JDK Runtime Environment.

B. Experiments and Results

The evaluation was conducted as follows. First, the success ratio and execution time is measured for increasing iterations, and then increasing population sizes are evaluated. Afterwards, the different weight functions are tested. And finally, experiments consisting of measuring increasing numbers of workflows are performed.

Figures 2 and 3 show the experiments with increasing iterations. The population size was set to 10, and the number of workflows was set to 300.

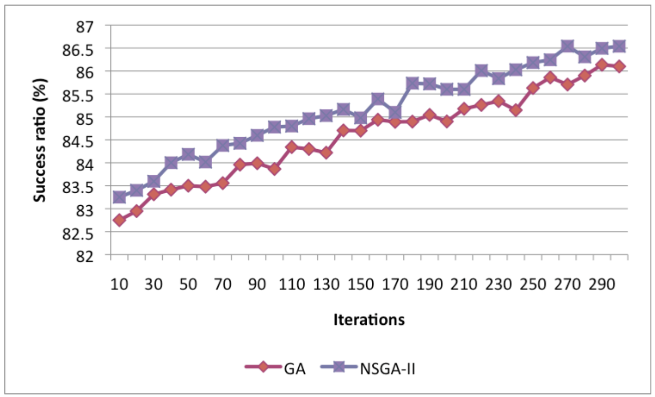

Figure 2. Success ratio with increasing iterations.

Figure 2 shows that the success ratio for both algorithms increases linearly. NSGA-II has the higher success ratio (roughly 1 %) than GA. However, as the execution time in Figure 3 reveals, NSGA-II has a steeper increase than GA showing that NSGA-II takes much longer than GA, in particular for larger numbers of iterations. For example, NSGA-II takes 107 milliseconds to run, whereas GA takes 64 milliseconds for 300 iterations.

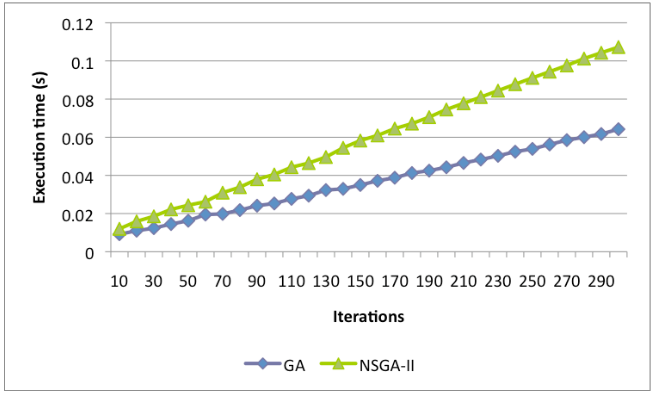

Figure 3. Execution time with increasing iterations.

Figures 4 and 5 show the success ratio and execution time for increasing population sizes. The number of iterations was set to 100, and the number of workflows was set to 300.

In Figure 4, we can observe that the curves of both algorithms, GA and NSGA-II, are crossing each other at around a population size of 30. First, NSGA-II has the higher success ratio, however after the population size of 30 is reached, GA continues to have higher scores.

Figure 5 shows the execution time confirming once again, that NSGA-II's performance is less than that of the GA.

Figures 6 and 7 investigated different fitness functions. Table 2 shows the different weight distributions used. The number of iterations was 100, the number of workflows was 300, and the population size was 100.

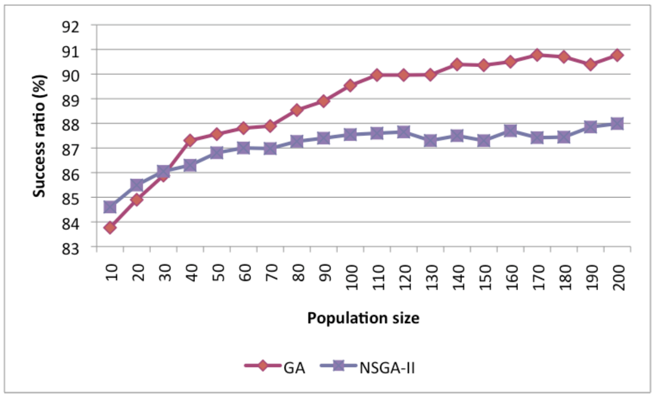

Figure 4. Success ratio for increasing population sizes.

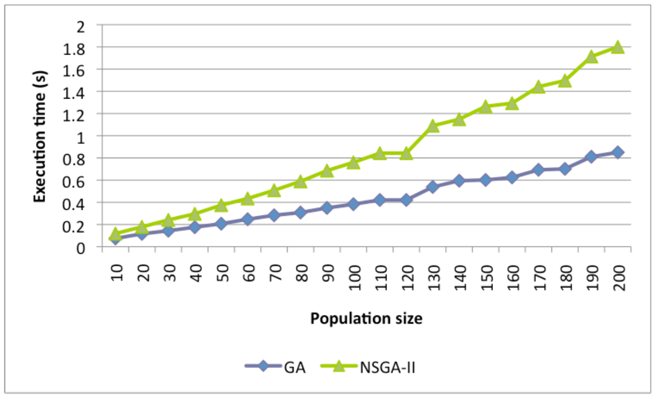

Figure 5. Execution times for increasing population sizes.

TABLE II. WEIGHT DISTRIBUTION FOR DIFFERENT FITNESS FUNCTIONS

	w_1	w_2	w_3	w_4
F1	0.25	0.25	0.25	0.25
F2	0.4	0.2	0.2	0.2
F3	0.2	0.4	0.2	0.2
F4	0.2	0.2	0.4	0.2
F5	0.2	0.2	0.2	0.4

Figure 6 shows the success ratio of the different weights used for the fitness functions. It can be seen that the GA algorithm is not affected by this, showing stable success ratios (within the variance of ±0.48% of the measurements) for all different fitness functions. Similar behavior is observed for the NSGA-II algorithm. No significant difference can be seen of the success ratios for the different fitness functions.

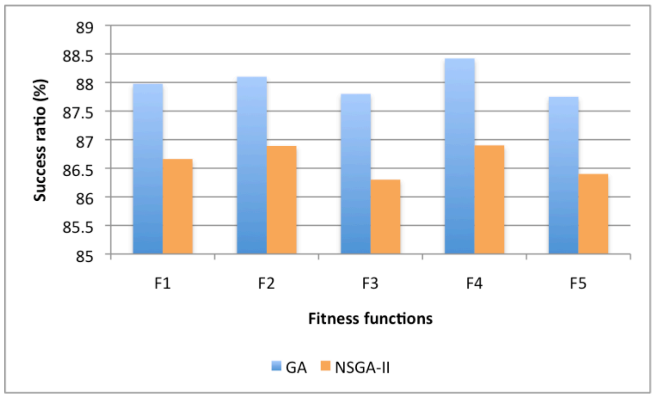

Figure 6. Success ratio for different fitness functions.

Figure 7 shows the execution times for varying fitness functions. It shows similar execution times for all fitness functions within the variance of the measurements.

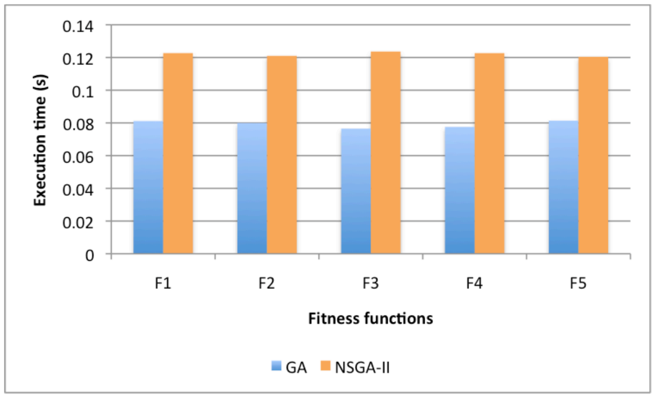

Figure 7. Execution time for different fitness functions.

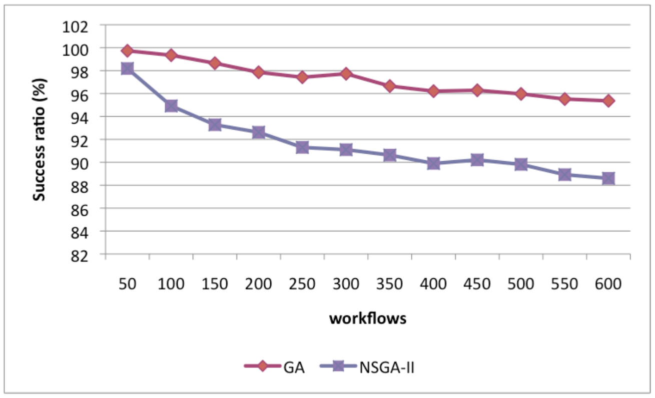

Figure 8. Success ratio for increasing workflows.

Figures 8 and 9 show the success ratio and execution time for increasing workflows. The number of iterations was set to 2000, and the population size was 100.

Figure 8 reveals that the success ratio of GA is higher than NSGA-II. GA almost achieving 100% for 500 workflows (99.7%), closely followed by NSGA-II scoring 98.1%. However, when the number of the workflows increases, the wider the gap between GA and NSGA-II becomes. At iteration 600, NSGA-II scores 95.4% and GA only 88.6%.

Figure 9 shows the execution time for increasing workflows with a linear trend of both algorithms. Once again, NSGA-II is revealing its higher computational cost.

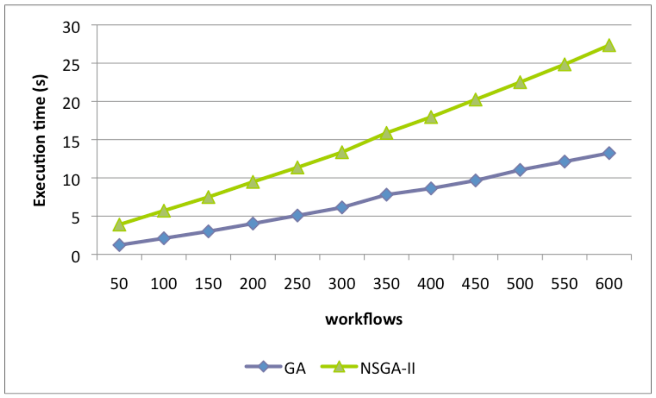

Figure 9. Execution time for increasing workflows.

V. Conclusion

Workflow composition is a very important issue for service-oriented environments. Past research had addressed this workflow composition problem with single-objective and multi-objective optimization approaches. Most of the research had employed the single-objective approach whereby the different objectives, i.e., QoS attributes, are aggregated by a weighted approach. Fewer research approaches used the multi-objective approach, whereby several solutions are produced by a set of Pareto solutions, which have equivalent quality to satisfy the service level agreements.

However, no comparison had been done investigating both approaches. Therefore, this paper addressed this by measuring the success ratio as well as the execution time of two implemented GA approaches. The first was a generic GA approach for the single-objective version, whereas the well-known NSGA-II algorithm was implemented as the multi-objective variant.

As can be seen by the experimental results, the NSGA-II algorithm has only slightly higher success rates for small populations sizes and a small number of iterations. However, overall the generic GA algorithm performs much better both in terms of success rate as well as execution time when using standard GA settings (population size of 100, number of iterations of 100), which also leads to higher success ratios.

Even in the case of different fitness functions with different weight distributions for each objective, i.e., QoS attribute, no significant difference can be seen between both approaches. Usually, multi-objective algorithms are employed if the weights of the different objectives cannot be estimated, however, in our case, where we have four conflicting QoS parameters, the user can easily assign different weight values. Nevertheless, the different fitness functions did not have any effect on the success ratio or execution times of both approaches.

A recommendation is to use a single-objective algorithm with an aggregated weighted approach, and not a multi-objective approach in particular considering the processing time of the optimization process in service-oriented environments. Also, the success ratio was much higher for the single-objective approach.

Future work will expand this line of research by taking the following constraints imposed by the real world setting into consideration. First of all, service invocations of a particular service are limited, and therefore, needs to be taken into account. Furthermore, failure of the service execution and re-composition needs to be addressed, and a solution needs to be implemented and tested.

Acknowledgment

This material is based on work supported by North Dakota EPSCoR and National Science Foundation Grant EPS-0814442.

References

[1] M.N. Huhns, M.P. Singh, Service-Oriented Computing: Key Concepts and Principles, IEEE Internet Computing 9(1), 75-81, 2005.

[2] L. Taher, H. El Khatib and R. Basha, A framework and QoS matchmaking algorithm for dynamic web services selection, Second International Conference on Innovations in Information Technology (IIT'05), Dubai, UAE, 2005.

[3] M. Comuzzi, B. Pernici, A framework for QoS-based Web service contracting, ACM Trans. Web, Vol. 3, No. 3. 2009.

[4] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, C.-H. Chen, Dynamic Web Service Selection for Reliable Web Service Composition, IEEE Transactions on Services Computing, pp. 104-116, 2008.

[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, 1(1), 2007.

[6] D. Ardagna and B. Pernici, Adaptive Service Composition in Flexible Processes, IEEE Transactions on Software Engineering, 33(6):369–384, 2007.

[7] D. Schuller, J. Eckert, A. Miede, S. Schulte, R. Steinmetz, QoS-Aware Service Composition for Complex Workflows, Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services, 2010.

[8] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q.Z. Sheng, Quality driven web services composition, Proceedings of the 12th international conference on World Wide Web, 2003.

[9] G. Canfora, M.D. Penta, R. Esposito, and M.L. Villani, An Approach for QoS-aware Service Composition based on Genetic Algorithms, Proceedings of Conference on Genetic and Evolutionary Computation, 2005.

[10] K. Yoon, C.L. Hwang, Multiple Attribute Decision Making: An Introduction. Sage Publishers, 1995.

[11] M.C. Jaeger and G. Muehl, QoS-based Selection of Services: The Implementation of a Genetic Algorithm, Proceedings of Conference on Communication in Distributed Systems, Workshop on Service-Oriented Architectures and Service-Oriented Computing, 2007.

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

[13] D.B. Claro, P. Albers, and J. Hao. Selecting Web Services for Optimal Composition, In IEEE International Conference on Web Services, Workshop on Semantic and Dynamic Web Processes, 2005.

[14] H. Wada, P. Champrasert, J. Suzuki, K. Oba, Multiobjective Optimization of SLA-Aware Service Composition, Congress on Services, pp. 368-375, 2008.

[15] M. Mitchell. An Introduction to Genetic Algorithms (Complex Adaptive Systems), The MIT Press, ISBN 0-262-63185-7, 1998.

[16] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, A comparison of genetic sequencing operators, In Rick Belew and Lashon Booker, editors, Morgan Kaufman, Proceedings of the Fourth International Conference on Genetic Algorithms, 69-76, San Mateo, CA, 1991.

[17] K. Deb. Multi-objective optimization using evolutionary algorithms, Wiley, Chichester, England, 2001.

[18] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, In Lecture Notes in Computer Science, volume 1917, 848-849, 2000.