Memetic Algorithm for Web Service Selection

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

ABSTRACT

Due to the changing nature of service-oriented environments, the ability to locate services of interest in such open, dynamic, and distributed environments has become an essential requirement. Current service-oriented architecture standards mainly rely on functional properties, however, service registries lack mechanisms for managing services' non-functional properties. Such non-functional properties are expressed in terms of quality of service (QoS) attributes. QoS for web services allows consumers to have confidence in the use of services by aiming to experience good service performance in terms of waiting time, reliability, and availability. This paper investigates the service selection process, and proposes two approaches; one that is based on a genetic algorithm, and the other is based on a memetic algorithm to match consumers with services based on QoS attributes as closely as possible. Both approaches are compared with an optimal assignment algorithm called the Munkres algorithm, as well as a Random approach. Measurements are performed to quantify the overall match score, the execution time, and the scalability of all approaches.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence, Problem Solving, Control Methods, and Search, Heuristic methods.

General Terms

Algorithms, Measurement, Performance.

Keywords

Evolutionary computing, genetic algorithms, memetic algorithms, munkres algorithm, consumer-provider matching, quality of service.

1. 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 from 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 quality of service (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].

The selection of an appropriate service for a particular task has become a difficult challenge due to the increasing number of web services offering similar functionalities. Therefore, research was conducted to investigate different approaches to address this problem. This paper introduces two service selection approaches based on evolutionary computing. One approach makes use of a genetic algorithm, whereas the other is based on a memetic algorithm.

BADS'11, June 14, 2011, Karlsruhe, Germany.

The paper is structured as follows: Section 2 introduces and discusses related work. In Section 3, the problem specification and the approaches and implementations are discussed. Section 4 describes the measurement setup and discusses the results, and Section 5 concludes this paper with a comparison analysis.

2. RELATED WORK

In [3], a personalized selection approach for web services by partitioning user profiles to support different steps of interaction with services using techniques to personalize each subsequent step is proposed. The algorithm features the expansion of service requests by user-specific demands and wishes. Services that do not match a certain profile are discarded on the fly and equally useful results of alternative services are compared with respect to strategies and preferences provided by the user.

Another web service selection scheme is proposed in [4]. The approach is based on user's requirement of the various non-functional properties and interaction with the system. The framework utilizes user preferences as an additional input to the selection engine and the system ranks the available services based on the requirement.

A model of web service configurations and associated prices and preferences using utility function policies is described in [5]. The approach takes ideas from multi-attribute decision theory to develop an algorithm for optimal service selection. This approach represents configurable web service offers and requests in an ontology, and it makes use of declarative logic-based matching rules with optimization methods, such as linear programming, in order to solve it.

A non-functional property-based service selection method, modifying the logic scoring preference method with ordered weighted averaging operators is introduced in [6]. The dynamic mechanism for evaluating metadata based on QoS criteria is proposed.

A multi-agent approach based on an architecture and programming model in which agents represent applications and services is presented in [7]. The agents support considerations of semantics and quality of service parameters. The agents interact and share information, in essence creating an ecosystem of collaborative service providers and consumers. The approach enables applications to be dynamically configured at runtime in a manner that continually adapts to the preferences of the participants. The agents are designed to use decision theory.

In [8], an approach is described that uses a tree structure, called quality constraint tree, to represent the requester's variety of requirements on QoS properties having varied preferences. The QoS broker architecture facilitates the requesters to specify their QoS requirements to select qualitatively optimal web services. The algorithm ranks the functionality of similar web services based on the degree of satisfaction of the requester's QoS requirements and preferences.

An efficient service selection scheme to help service requesters select services by considering two different contexts, single QoS-based service discovery and QoS-based optimization of service composition, is introduced in [9]. Based on QoS measurement metrics, the approach proposes multiple criteria decision-making and integer-programming approaches to select the optimal service.

Research regarding the optimization of service selection to determine the optimal selection of services based on a measurable QoS metric is proposed by the following approaches.

In [10], a broker-based architecture is proposed with a technique that models the problem in two ways. A combinatorial model and a graph model are proposed. The combinatorial model defines the problem as a multi-dimensional multi-choice 0-1 knapsack problem. The graph model defines the problem as a multi-constraint optimal path problem. Both models are studied and compared with each other.

The service selection algorithm proposed in [11] investigates the problem of composite web service selection. A utility function is proposed to evaluate all QoS parameters of each service based on the definition given in [10]. A multi-dimensional QoS composite web service is mapped to the multi-dimensional multi-choice knapsack. A fast heuristic algorithm is proposed for solving the selection problem.

Evolutionary computing has also been introduced to the service selection problem, in particular for workflow problems, using a genetic algorithm (GA) approach.

A GA approach for the selection of services using QoS requirements is introduced in [12]. A simple GA algorithm was implemented and tested in a simulation environment called SENECA.

Another GA approach with a quick convergence method is proposed in [13]. In particular, the quickly convergent population diversity handling GA uses an enhanced initial population policy and an evolution policy based on population diversity and a relation matrix coding scheme. The integration of the two policies overcomes shortcomings resulting from the random nature of GA, such as slow convergence, large variations among running results, soaring overhead along with increasing sizes of service compositions.

The aim of the research provided in this paper has a slightly different focus. First of all, only single service requests are investigated, and it is envisioned that the necessity of service selection support will increase in future, not only because the number of service-oriented applications are increasing, but also more and more services with similar functionality are becoming available on the web. Therefore, a robust, time-efficient and scalable assignment algorithm is needed to perform the task of service selection. One optimal algorithm, known as the Munkres algorithm, has a time complexity of $O(n^3)$, and therefore, does not scale well with increasing numbers of consumers and providers. Thus, approximate algorithms are necessary, which on one hand provide an optimized assignment, and on the other hand scale closely to linear with increasing numbers of consumer-provider pairs.

3. SERVICE SELECTION APPROACHES

The problem of service selection on the web consists of having an efficient algorithm that can match multiple service consumers and service providers efficiently, while optimizing multiple objectives (QoS parameters). The problem is twofold: firstly, multiple clients requesting similar services should be satisfied, and secondly, the assignment process of the service consumers and the service providers should be optimized. Please note that one consumer can only be matched with one provider.

The QoS criteria in the context of services are execution price, execution time, reliability, reputation, and availability. The values of these QoS parameters range between 0 to 1. Each consumer provides the QoS values based on its requirement of how the request must be executed, and each service provides the value based on its task execution capability. The service provider has a value for each QoS parameter. The service consumer requests a service provider specifying an upper and lower value for each QoS parameter, whereby for some QoS attributes the lower or upper bound is preferred. In particular, the lower bound is preferred for execution price and execution time, and the upper bound is preferred for reliability, reputation and availability.

In order to calculate how good and close matches are, the following equations are used (keeping in mind that several consumers are matched with several providers simultaneously):

$$v_i = \begin{cases} 0 & \text{if } p_i \geq cu_i \text{ or } p_i < cl_i \\ 1 - \dfrac{cu_i - p_i}{cu_i - cl_i} & \text{if lower bound preferred} \\ 1 - \dfrac{p_i - cl_i}{cu_i - cl_i} & \text{if upper bound preferred} \end{cases} \quad (1)$$

$$m = \frac{1}{5}\sum_{i=1}^{5} v_i \quad (2)$$

$$o = \frac{1}{n}\sum_{j=1}^{n} m_j \quad (3)$$

whereby o is the overall match score of the problem (and therefore the fitness function for the algorithms), m is the match score for a consumer-provider pair, v_i is the match value, cu_i and cl_i are the upper and lower value of the consumer respectively, p_i is the value of the provider, i is the QoS parameter, and j is the service number.

Since we have several service consumers and equally numbered service providers, the aim is to match the consumer–provider pairs as closely as possible using a GA and a Memetic Algorithm (MA) approach.

3.1 Munkres Algorithm

The combinatorial optimization algorithm that solves the assignment problem in polynomial time is referred to as the Hungarian algorithm. The algorithm was developed by Harold Kuhn in 1955 [14,15]. Kuhn named it "Hungarian method" because two Hungarian mathematicians, Denes Koenig and Jeno Egervary, were the first who worked on the algorithm. In 1957 the algorithm was reviewed by James Munkres and he observed its (strongly) polynomial behavior, and therefore the algorithm was given the name Kuhn-Munkres algorithm or Munkres assignment algorithm [16,17]. The time complexity of the original algorithm was $O(n^4)$, however, it was later modified by Edmonds and Karp (and independently by Tomizawa) to achieve a $O(n^3)$ running time [18].

The Munkres algorithm is used to serve as a benchmark for the service selection as it is an optimal algorithm. The assignment problem, as formally defined by Munkres, is the following [16]:

"Let rij be a performance ratings for a man Mi for job Ji. A set of elements of a matrix are said to be independent if no two of them lie in the same line ("line" applies both to the rows and the columns of a matrix). One wishes to choose a set of n independent elements of the matrix (rij) so that the sum of the element is minimum."

Similarly, the problem of matchmaking can be defined as an consumer-provider matrix, representing the match scores of each consumer with every other provider. The match score matrix is the matrix represented in Equation (3), where each element of the matrix represents the match score for an individual consumer-provider pair. The Munkres algorithm works on this matrix, to assign the consumer requests to providers, as to achieve an overall maximum total match score. Please note that one consumer can only be matched with one provider.

An implementation developed by Nedas in Java, which is freely available at [19] was slightly adapted and used to provide the benchmark for the service selection investigation, as it provides an optimal assignment of consumer and provider pairs.

3.2 Genetic Algorithm (GA)

GA is a global optimization algorithm that models natural evolution [20]. In GA, individuals form a generation. An individual corresponds to one match. The match is implemented as a vector, which is also referred to as a chromosome. Dimensions in the vector correspond to providers, and values correspond to consumers. Therefore, if the vector has value 3 at its 5th position (dimension), consumer 3 is matched with provider 5. Every number representing a consumer can only be at one position in the vector, otherwise, the vector represents a non-valid match.

At the beginning, the first population is randomly initialized. After that, the fitness of the individuals is evaluated using the fitness function (Equations (1)–(3)).

After the fitness is evaluated, individuals have to be selected for paring. The selection method used is tournament selection. Always two individuals are paired, resulting in an offspring of two new individuals. In the pairing phase, a random crossover mask is used, i.e. the positions (dimensions) for which crossover occurs are selected randomly.

If crossover occurs at certain positions (dimensions), individuals that are mated exchange their values at that position and the resulting individuals are used as offspring. The crossover has to make sure that the offspring presents a valid match. Therefore, if two values are exchanged, other positions in the two match vectors are usually effected as well.

The offspring faces mutation with a certain low probability. After mutation, the fitness of the offspring is calculated. Then, either all individuals from the last generation compete against the whole offspring, or the offspring only compete with its corresponding parents.

In this implementation, all individuals from the old generation compete with all individuals in the new generation. In order to implement this, all individuals are ordered by their fitness score, using a non-recursive advanced quicksort algorithm, which after sorting truncates the lower half. After the new generation is selected, the GA will start over, and continue with parent selection and crossover until a certain number of iterations is reached.

3.3 Memetic Algorithm (MA)

Evolutionary algorithms are not well suited for fine-tuning the search, in particular in complex combinatorial spaces, and therefore, researchers have developed hybridization methods to

overcome this problem and to improve the efficiency of the search [21].

The combination of using evolutionary algorithms as well as local search techniques was named Memetic Algorithms (MAs). MAs are basically an extension of evolutionary algorithms that apply a separate process to refine solutions by improving the fitness of the individuals with methods such as hill-climbing or simulated annealing.

MAs were inspired by the models of adaptation in natural systems, in particular the combination of the evolutionary adaptation of a population with individual learning. GAs on the one hand and local search on the other hand, are captured within MAs, thus rendering a methodology that balances well between generality and problem specificity. The name of MAs was inspired by Richard Dawkins' concept of a meme, which represents a unit of cultural evolution that can exhibit local refinement [22]. A meme represents a learning or development strategy [23].

Memetic algorithms are also know as Hybrid Evolutionary Algorithms [24], Baldwinian Evolutionary Algorithms [25], Lamarckian Evolutionary Algorithms [26], Cultural Algorithms or Genetic Local Search. All techniques combine local search heuristics with the evolutionary algorithms' operators. Combinations with constructive heuristics or exact methods may also be considered within this class of algorithms. In this research, we apply an exact method for the local search.

For some problem domains, MAs have been shown to be both more efficient and more effective than traditional evolutionary algorithms with regards to requiring orders of magnitude fewer evaluations to find optima, and identifying higher quality solutions. In particular, for many combinatorial optimization problems, where large instances have been solved to optimality, and where other meta-heuristics have failed to produce comparable results, such as the quadratic assignment problem and the traveling salesman problem**,** MAs have proven themselves to be very effective [23].

For the problem of matchmaking, the following local search technique is applied after the crossover and mutation phases have finished. For 10% of the population size, one individual is randomly chosen, and thereof 10% of the consumer-provider pairs are also randomly chosen to perform the Munkres algorithm on this selection. If after applying the Munkres algorithm an improvement is achieved, then that particular individual is updated with the optimized match pairs and the next iteration continues.

4. EXPERIMENTS AND RESULTS

All three approaches as introduced in the previous section were implemented. In addition, the random approach, which randomly matches consumers with providers, is used for comparison reasons.

Experiments were designed to measure the overall match score and the execution time of all approaches. The GA and MA algorithms were further analyzed with regard to the number of iterations and the population size used. All measurement points shown are average results taken from 30 runs in order to guarantee an equal distribution and statistical correctness. The data sets for the consumers and providers were randomly generated and solved by Munkres, GA, MA, and the Random approach. All match scores shown are normalized with respect to the Munkres algorithm.

The following parameters have been chosen due to their superior performance on the service selection problem, balancing between accuracy and execution time, also with regards to the scalability analysis.

For the GA and MA, the parameters were set to: population size = 400, iteration = 10, crossover probability = 60%, mutation probability = 0.5%, size of the tournament selection = 10, and number of positions that are selected for crossover = 10%. The MA has some specific parameters set for the local search component, which are individuals selected set to 10%, and the consumer-provider pair selection was set to 10%.

The experiments were conducted on an Intel Core 2 Duo (2.5GHz, 3MB L2 cache) running the Java Version 1.6.0 OpenJDK Runtime Environment.

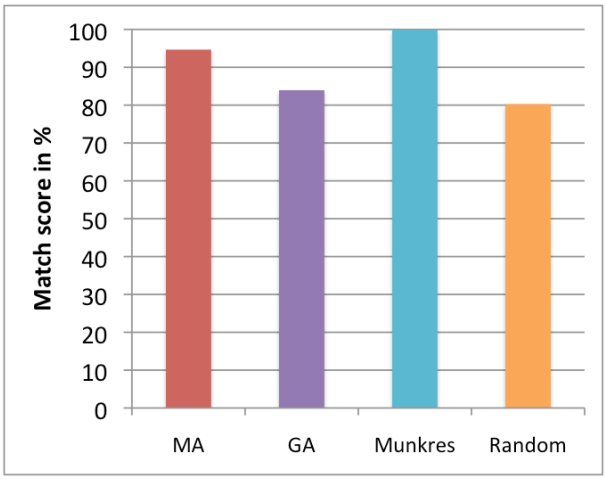

Figure 1. Match score of all algorithms

Figure 1 shows the match score of all four algorithms, using 500 consumer-provider pairs. As expected, the Munkres algorithm achieves a match score of 100%, whereby the MA comes quite close with 94.7%, followed by the GA with 83.9%, and the Random approach only scoring 80.3%.

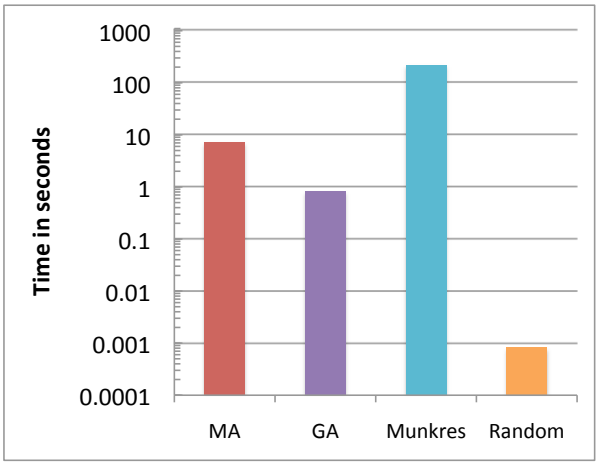

Figure 2. Execution time of all algorithms

Figure 2 shows the execution time in seconds of all four algorithms, again using 500 consumer-provider pairs (i. e., 500 consumers and 500 providers).

As can be seen, Munkres, being an optimal algorithm, has the largest execution time, followed by the MA algorithm, then the GA algorithm, and at last the Random approach. Please note that the y-axis is in logarithmic scale. The execution time for Munkres was 210.7 seconds, for the MA it was 7.0 seconds, the GA ran 0.8 seconds, and the Random approach needed not even 1 second to run.

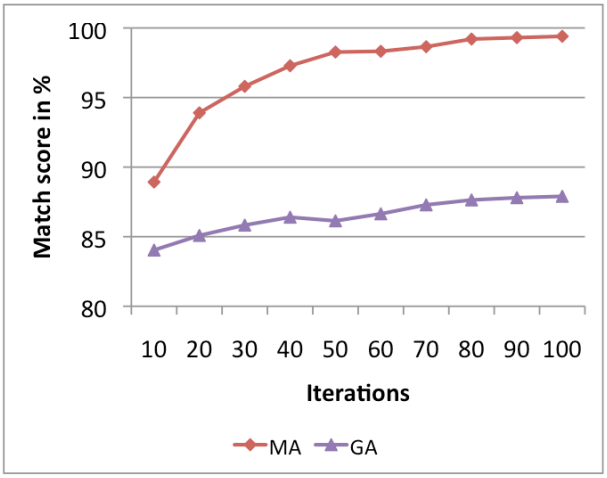

Figure 3. Match score vs. number of iterations of the GA and MA algorithms

For the next experiments only both evolutionary algorithms are compared in terms of increasing numbers of iterations, as well as increasing population sizes.

Figure 3 plots the match score against the number of iterations. As can be seen, the GA algorithm slowly but steadily increases the match score with every iteration, however, the MA algorithm shows a large increase at the beginning and is then flattening out with increasing numbers of iterations when approaching a match score of 100%. At iteration 100 the match score of MA is 99.0% and the match score of GA is 87.8%. The reason for the larger increase at the beginning is due to the local search component of the MA algorithm.

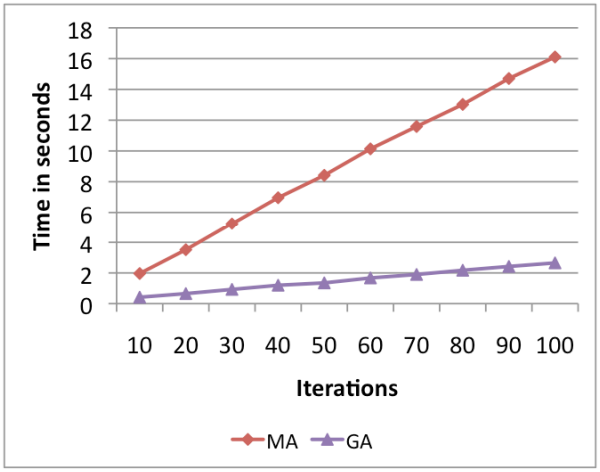

Figure 4. Time vs. number of iterations of the GA and MA algorithms

Figure 4 shows a linear increase in time for increasing numbers of iterations for the GA and MA algorithms, whereby the MA algorithm shows the larger increase, due to the additional time needed for the local search.

Figures 5 and 6 show the match score and execution time for both evolutionary algorithms when increasing the population size.

In Figure 5, a slight increase in match score for both algorithms is shown, whereby MA achieves a match score of 91.4%, and GA scoring 84.7% for a population size of 2000.

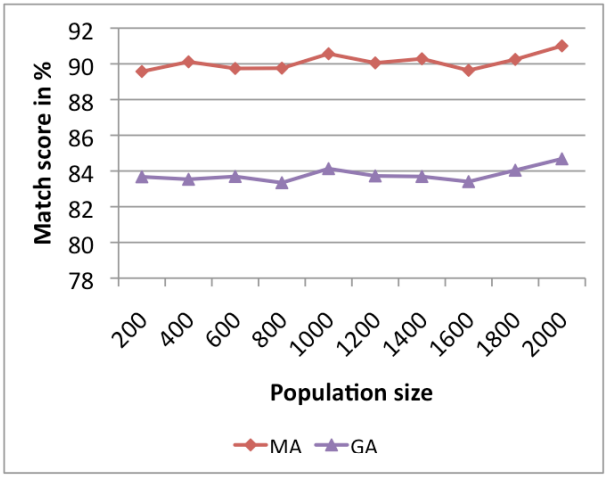

Figure 5. Match score vs. population size of the GA and MA algorithms

Figure 6 shows, as expected, a linear increase in time for increasing population sizes, showing a larger increase for the MA algorithm.

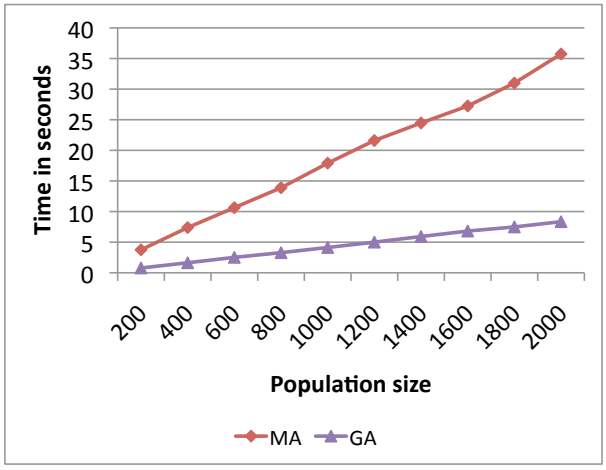

Figure 6. Time vs. population size of the GA and MA algorithms

Since this investigation is primarily concerned with the scalability of the algorithms, the match score and execution time with increasing numbers of consumer-provider pairs is explored, showing the results in Figures 7 and 8.

The consumer-provider pairs were increased up to 600 in steps of 50. Figure 7 displays the match score for increasing numbers of consumer-provider pairs showing relatively stable match score values for all algorithms, however, indicating a slight decrease with larger consumer-provider pairs.

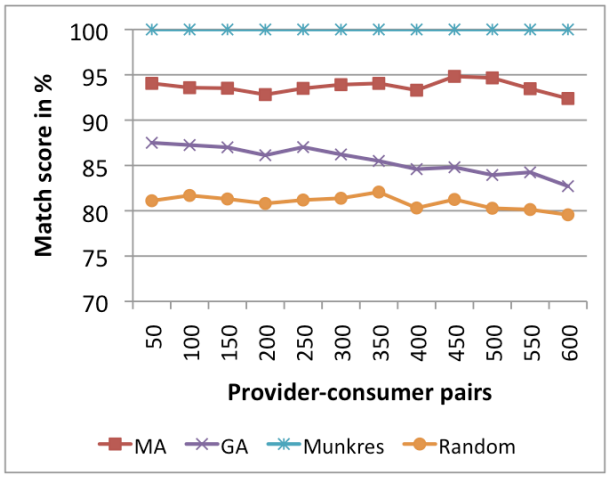

Figure 7. Scalability of algorithms in terms of match score

Figure 8 shows the execution time of all algorithms, highlighting that the Munkres algorithm does scale in a cubic fashion, whereas the evolutionary algorithms and the Random approach scale linearly.

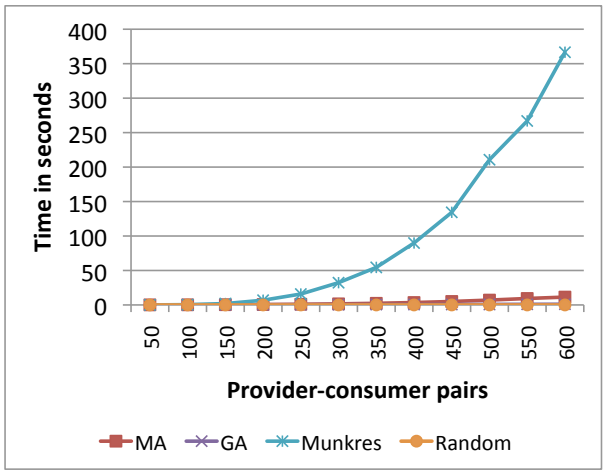

Figure 8. Scalability of algorithms in terms of execution time

5. CONCLUSION

This paper investigated four approaches for the selection of services, and in particular the selection of multiple consumers and providers based on five QoS attributes. The scenario was envisioned, that in future the number of services will increase, and therefore, an efficient algorithm is necessary to provide a good selection scalability.

The Munkres algorithm provides an optimal assignment, however, it has a computational time complexity of $O(n^3)$, and thus, does not scale very well. Therefore, the GA algorithm was implemented to see whether good match scores could be achieved. However, preliminary measurement results showed that GA's match score was right in the middle, between the 100% achieved by the Munkres algorithm, and the 80% achieved by the random approach. In order to achieve a higher match quality, the MA algorithm was implemented and evaluated. As shown by the experimental results, the MA shows a good balance between match score and execution time by achieving a match score of 94.7%; the GA scores 83.9%, and the Random approach only scores 80.3% for 500 consumer-provider pairs. In terms of execution time and scalability, the MA algorithm, even though needing extra time to perform the local search, when compared to the genetic algorithm, the MA still shows a linear behavior and is not much worse than the GA (for 600 consumer-provider pairs: Munkres: 366.6 seconds, MA: 11.4 seconds, and GA: 1.1 seconds).

Future work will follow three directions. First of all, since the consumer-provider pairs were fairly similar in terms of range, the effect of larger variations in the consumer-provider pairs will be investigated, likely achieving even higher match scores. Secondly, a non-dominated sorting GA, as well as a Particle Swarm Optimization approach will be investigated and compared to the current approaches. Thirdly, different selection matching functions will be experimented with.

6. REFERENCES

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

[2] Taher, L., El Khatib, H. 2005. A framework and QoS matchmaking algorithm for dynamic web services selection, Proceedings of the 2nd International Conference on Innovations in Information Technology (IIT'05).

[3] Balke, W.T., Wagner, M. 2003. Towards personalized selection of web services, In Proceedings of the Int. World Wide Web Conf. (WWW).

[4] Badr, Y., Abraham, A., Biennier, F., Grosan, C. 2008. Enhancing Web Service Selection by User Preferences of Non-functional Features, Proceedings of the 2008 4th international Conference on Next Generation Web Services Practices.

[5] Lamparter, S., Ankolekar, A., Studer, R., Grimm, S. 2007. Preference-based selection of highly configurable web services, Proceedings of the 16th international conference on World Wide Web (WWW).

[6] Yu, H.Q., Reiff-Marganiec, S. 2008. A Method for Automated Web Service Selection, Proceedings of the 2008 IEEE Congress on Services.

[7] Maximilien, E.M., Singh, M.P. 2004. Toward autonomic web services trust and selection. In Proceedings of the 2nd international Conference on Service Oriented Computing (ICSOC).

[8] D'Mello, D.A., Ananthanarayana, V.S. 2010. Dynamic selection mechanism for quality of service aware web services, Journal of Enterprise Information Systems, Taylor & Francis, vo. 4, no. 1, pp. 1751-7575.

[9] Huang, A.F., Lan, C., Yang, S.J. 2009. An optimal QoS-based Web service selection scheme. Journal of Inf. Sci. vol. 179, no. 19, pp. 3309-3322.

[10] Yu, T., Zhang, Y., Lin, K. 2007. Efficient algorithms for Web services selection with end-to-end QoS constraints, ACM Transaction Web, vol. 1, no. 1, pp. 1-26.

[11] Wang, R., Chi, C., Deng, J. 2009. A Fast Heuristic Algorithm for the Composite Web Service Selection, Proceedings of the Joint international Conferences on Advances in Data and Web Management.

[12] Jaeger, M.C., Mühl, G. 2007. QoS-based selection of services: The implementation of a genetic algorithm,

Proceeding of KiVS (Kommunikation in Verteilten Systemen) in Workshop: Service-Oriented Architectures und Service Oriented Computing.

[13] Ma, Y., Zhang, C. 2008. Quick convergence of genetic algorithm for QoS-driven web service selection, Journal of Computer Networks, vol. 52, no. 5, pp. 1093-1104.

[14] Kuhn, H.W. 1955. The Hungarian method for the assignment problem, Naval Research Logistics, 52(1).

[15] Kuhn, H.W. 1955. The hungarian method for solving the assignment problem, Naval Research Logistics Quarterly, 2:83.

[16] Munkres, J. 1957. Algorithms for the Assignment and Transportation Problems, Journal of the Society for Industrial and Applied Mathematics, 5:32.

[17] Bourgeois, F., Lassalle, J.C., 1971. An extension of the munkres algorithm for the assignment problem to rectangular matrices, Commun. ACM, 14(12).

[18] Wikipedia, Hungarian Algorithm, last retrieved March 2011, http://en.wikipedia.org/wiki/Hungarian_algorithm.

[19] Nedas, K. 2009. Munkres' (Hungarian) Algorithm, Java implementation, last retrieved on March 2009 from http://konstantinosnedas.com/dev/soft/munkres.htm.

[20] Holland, J.H. 1975. Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor.

[21] Wolpert, D., Macready, W. 1997. No free lunch theorems for optimization, IEEE Trans. Evol. Comput., vol. 1, no. 1, pp. 67-82.

[22] Dawkins, R. 1976. The Selfish Gene, New York: Oxford Univ. Press.

[23] Krasnogor, N., Smith, J. 2005. A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues, IEEE Trans. On Evolutionary Computation, vol. 9, no. 5.

[24] Vazquez, M., Whitley, L. 2000. A hybrid genetic algorithm for the quadratic assignment problem, Proc. Genetic Evol. Comput. Conf., D, pp. 135-142.

[25] Ku, K.,Mak, M. 1998. Empirical analysis of the factors that affect the Baldwin effect, Lecture Notes in Computer Science, Parallel Problem Solving From Nature, pp. 481-490.

[26] Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K., Olson, A.J. 1998. Automated docking using a lamarkian genetic algorithm and an empirical binding free energy function, Journal Comput. Chem., vol. 14, pp. 1639-1662.