Web Service Selection using Particle Swarm Optimization and Genetic Algorithms

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

Thomas Schoene
Department of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada
ths299@mail.usask.ca

***Abstract*— Current service-oriented architecture standards mainly rely on functional properties, however, service registries lack mechanisms for managing services' non-functional properties. These non-functional properties are expressed in terms of quality of service (QoS) attributes, which gives consumers assurance and confidence to use the services, as consumers aim to experience a good service performance, e.g. low waiting time, high reliability, and availability. This paper investigates service selection, and proposes two approaches; one which is based on a genetic algorithm and the other is based on a particle swarm optimization approach 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. Measurements are performed to quantify the overall match score, the execution time, and the scalability.**

Quality of service; evolutionary computing; swarm intelligence; Munkres algorithm; matching problem

I. INTRODUCTION

Service-oriented computing is a computing paradigm that provides an environment in which services are loosely connected and interact with one another, as well as creates dynamic business processes and applications. Services are the fundamental building blocks in service-oriented environments and support rapid, low-cost development of distributed applications in heterogeneous environments. Service-oriented architectures enable service discovery, integration, and use by enabling application developers to overcome many distributed computing challenges.

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, which gives consumers assurance and confidence to use the services. Service registries host hundreds of similar web services, which make it difficult for the service consumers to choose from, given that the selection is only based on the functional properties. 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.

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.

II. RELATED WORK

A model of web service configurations and associated prices and preferences using utility function policies is described in [1]. 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 uses declarative logic-based matching rules with optimization methods such as linear programming for solving it.

A non-functional property-based service selection method, modifying the logic scoring preference method with ordered weighted averaging operators is introduced in [2]. The dynamic mechanism for evaluating metadata based on QoS criteria is proposed.

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 [3], 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 [4] 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 [3]. 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 [5]. A simple GA algorithm was implemented and tested in a simulation environment SENECA.

Another GA approach with a quick convergence method is proposed in [6]. 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 size of service compositions.

The aim of the research 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 service-oriented applications is increasing, but also more and more services with similar functionality become 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.

III. 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 from 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 proposed (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.

As 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 particle swarm optimization (PSO) approach.

A. Discrete Guaranteed Convergence Particle Swarm Optimization Algorithm: DGC-PSO

PSO, as introduced in [7], is a swarm based global optimization algorithm. It models the behavior of bird swarms searching for an optimal food source. The movement of a single particle is influenced by its last movement, its knowledge, and the swarm's knowledge.

PSO's basic equations are:

$$x_i(t+1) = x_i(t) + v_{ij}(t+1) \quad (4)$$

$$v_{ij}(t+1) = w(t)v_{ij}(t) + c_1 r_{1j}(t)\left(xBest_{ij}(t) - x_{ij}(t)\right) + c_2 r_{2j}(t)\left(xGBest_j(t) - x_{ij}(t)\right) \quad (5)$$

where x represents a particle, i denotes the particle's number, j the dimension, t a point in time, and v is the particle's velocity. *xBest* is the best location the particle ever visited (the particle's knowledge), and *xGBest* is the best location any particle in the swarm ever visited (the swarm's knowledge). w is the inertia weight and used to weigh the last velocity, c_1 is a variable to weigh the particle's knowledge, and c_2 is a variable to weigh the swarm's knowledge. r_1 and r_2 are uniformly distributed random numbers between zero and one. PSO is usually used on continuous and not discrete problems. In order to solve the discrete assignment problem using the PSO approach, several operations and entities have to be defined. This implementation follows the main ideas of the implementation for solving the traveling salesman problem as described in [8]. First, a swarm of particles is required. A single particle represents a match, i.e., every particle's position in the search space must correspond to a possible

match. The match, that is the position, is implemented as a vector. 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 has to be unique, otherwise, the vector represents a non-valid match.

Velocities are implemented as lists of changes that can be applied to a particle (its vector) and will move the particle to a new position (a new match). Changes are exchanges of values, i.e., an entity containing two values that have to be exchanged within a vector. This means that any occurrence of the first value is replaced by the second value, and any occurrence of the second is exchanged by the first value. Further, minus between two matches (particles), multiplication of a velocity with a real number, and the addition of velocities have to be defined. Minus is implemented as a function of particles. This function returns the velocity containing all changes that have to be applied to move from one particle to another in the search space. Multiplication randomly deletes single changes from the velocity vector, if the multiplied real number is smaller than one. When a velocity is added to another velocity, the two lists containing the changes will be concatenated.

The PSO implemented uses guaranteed convergence, which means that the best particle is guaranteed to search within a certain radius, implying that the global best particle will not get trapped in a local optima. This discrete guaranteed convergence algorithm is referred to as DGC-PSO from now on.

Configurable parameters in the implementation include numbers of particles (size of the swarm), number of iterations, c_1 (the weighting of the local knowledge), c_2 (the weighting of the global knowledge), w (the weighting of the last velocity), radius (defines the radius in which the global best particles searches randomly), global best particle swarm optimization (determines whether global best particle swarm or local best particle swarm optimization is used), and neighborhood size (defines the neighborhood size for local best particle swarm optimization).

B. Genetic Algorithm with Elitism: GA-E

GA is a global optimization algorithm that models natural evolution [9]. In GA, individuals form a generation. An individual (similar to a particle in the PSO) 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 present 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. After the new generation is selected, the GA will start over, and continue with parent selection and crossover. The implemented GA algorithm with elitism is referred to as GA-E from now on.

Configurable parameters in the implementation include number of iterations, 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.

C. Munkres Algorithm

The Hungarian algorithm is a combinatorial optimization algorithm and solves the assignment problem in polynomial time. The algorithm was developed by Harold Kuhn in 1955 [10,11], who named it "Hungarian method" since the algorithm was mainly based on the earlier works of two Hungarian mathematicians (Denes Koenig and Jeno Egervary).

In 1957 James Munkres reviewed and enhanced the algorithm and it has since been known also as Kuhn-Munkres algorithm or Munkres assignment algorithm [12,13]. The original algorithm was $O(n^4)$, however, Edmonds and Karp, and independently Tomizawa noticed that it can be modified to achieve a $O(n^3)$ running time.

An implementation developed by Nedas in Java, which is freely available at [14] was slightly adapted and used to provide the benchmark for the service selection investigation since it provides the optimal assignment of consumer and provider pairs.

IV. Experiments and Results

All three approaches as introduced in the previous section were implemented using Java. Experiments were designed to measure the overall match score and the execution time of all approaches. The DGC-PSO and GA-E algorithms were further analyzed with regard to the number of iterations and the number of particles/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 consumers and providers were randomly generated and solved by Munkres, DGC-PSO, and GA-E. All match scores shown are normalized with respect to the Munkres algorithm. For both, the DGC-

PSO and GA-E approaches, parameters have been optimized in two stages. In the first stage, values were optimized to yield high match scores. In the second stage, parameters have been optimized to yield fast execution times.

Preliminary results suggested that the number of iterations and number of particles (individuals) should be scaled with the number of consumer-provider pairs to achieve stable match scores for varying numbers of consumer-provider pairs, i.e., larger matches. The following scaling factor was used: $s_f = \frac{n_T}{n_B}$, where n_T corresponds to the number of consumer-provider pairs tested, and n_B corresponds to the number of consumer-provider pairs for which the basic time defining parameters have been set to yield a certain execution time. In this implementation, the values of the parameters that influence the execution time the most (number of iterations and number of particles/individuals) were set to yield half the execution time of the Munkres algorithm for 200 consumer-provider pairs. Therefore, n_B is set to 200. Furthermore, the parameters for both DGC-PSO and GA-E were optimized for 200 consumer-provider pairs.

The following parameters have been chosen due to their superior performance on the service selection problem. For the DGC-PSO, the parameters were set to: numbers of particles = 100, number of iterations = 140, c_1 = 0.5, c_2 = 1.8, w = 0.05, radius = 5.0 and using global best PSO. The GA-E settings were: population size = 150, number of individuals = 250, mutation probability = 0.2, crossover probability = 1.0, size of the tournament selection = 4, and number of positions that are selected for crossover (in percent) = 0.1.

The experiments were conducted on one core of a AMD Turion X2 Dual-Core Mobile RM-72 (2.1GHz, 512KB L2 Cache) on a Toshiba laptop with 2836 MB DDR2 memory using a Java Version 1.6.0 OpenJDK Runtime Environment on an Ubuntu 9.10 with runtime parameters set to: "–Xms512 –Xmx1024m –server". The normalized match score of all three algorithms measured, using 200 consumer-provider pairs are 1, 0.782, 0.761 for the Munkres DGC-PSO and GA-E respectively. As expected, the Munkres algorithm achieves an optimal match score of 1, whereby both evolutionary algorithms achieve match scores close to 0.8.

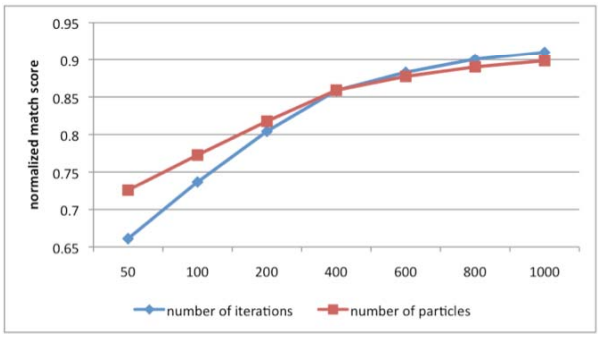

Figure 1. Normalized match score vs. number of iterations / particles of DGC-PSO algorithm.

The execution time in seconds of all three algorithms, again using 200 consumer-provider pairs are as follows: Munkres: 3547 ms, DGC-PSO: 1938 ms, and GA-E: 1875 ms. As can be seen, Munkres, being an optimal algorithm, has the largest execution time, followed by the DGC-PSO algorithm and the GA-E algorithm.

Figure 1 shows the normalized match score of the DGC-PSO algorithm for increasing numbers of iterations and increasing numbers of particles respectively. The distribution shows an early significant increase of the match score, with a smaller increase for larger numbers of iterations and particles respectively.

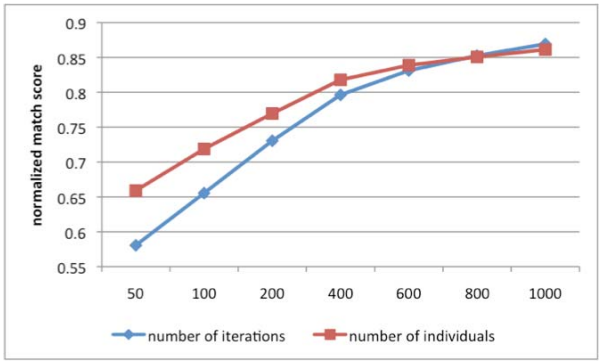

Figure 2. Normalized match score vs. number of iterations / individuals of GA-E algorithm.

A similar trend can be observed for the GA-E algorithm in Figure 2. A large increase of the match score for smaller number of iterations and individuals can be seen, followed by a smaller increase for larger numbers of iterations and individuals.

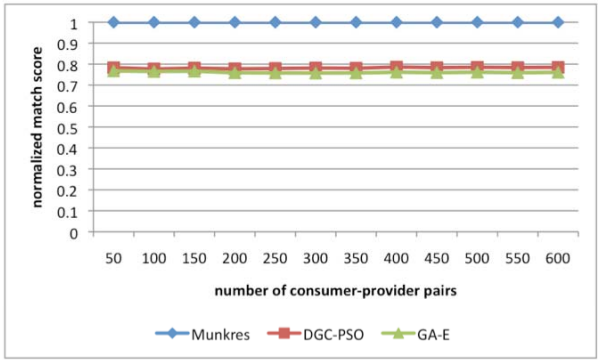

Figure 3. Scalability of algorithms – for scaled number of iterations and number of particles / individuals times s_f.

However, this investigation is primarily concerned with the scalability of the algorithms, in particular, the observation of the match score and execution time with increasing numbers of consumer-provider pairs. Figure 3 to 6 display the results using different scaling methods. Figure 3 shows the normalized match score of all algorithms scaling the number of iterations as well as the number of particles/individuals with s_f.

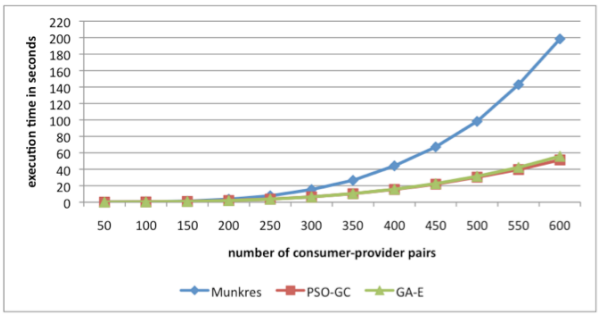

Figure 4. Scalability of algorithms – for scaled number of iterations and number of particles / individuals times s_f.

Figure 4 shows the execution time of all algorithms scaling the number of iterations as well as the number of particles/individuals with s_f showing that the Munkres algorithm does not scale as well as the evolutionary algorithms.

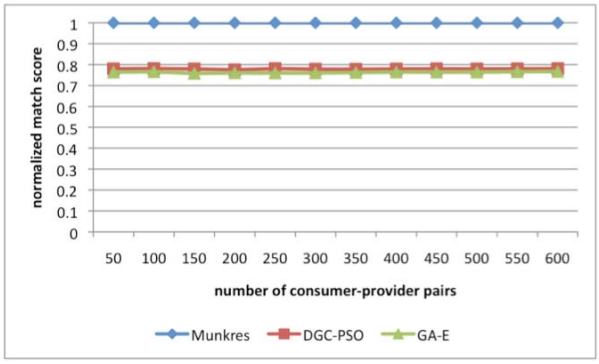

Figure 5. Scalability of algorithms – for scaled number of iterations times $s_f^{1.75}$.

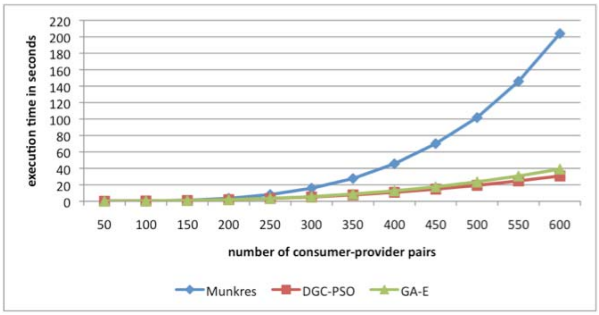

Figure 6. Scalability of algorithms – for scaled number of iterations times $s_f^{1.75}$.

Figure 5 shows the normalized match score of all algorithms scaling the number of iterations with $s_f^{1.75}$. This figure shows similar values (DGC-PSO: 0.780 / GA-E: 0.763) compared to Figure 5 (DGC-PSO: 0.782 / GA-E: 0.761).

Figure 6 shows the execution time of all algorithms scaling the number of iterations with $s_f^{1.75}$. It can be seen, comparing this figure with Figure 4, that the execution times are smaller for larger numbers of consumer-provider pairs, thus the scaling is improved.

V. CONCLUSION

This paper investigated three approaches for the selection of services, and in particular the selection of multiple consumers and providers based on five QoS attributes. 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 two approaches based on evolutionary algorithms, the DGC-PSO and GA-E algorithm were developed and implemented. The DGC-PSO and GA-E approaches matched the consumers with services based on the five QoS attributes yielding relatively good match scores while performing the matchmaking in a reasonable amount of time. Experiments showed that both algorithms provided an acceptable overall match score of close to 0.8 (DGC-PSO: 0.782 and GA-E: 0.761), and it only took approximately one sixth of the time for single scaling (compare Figure 5 and 6) and one fourth of the time for double scaling (compare Figure 3 and 4) compared to the Munkres algorithm for a match of 600 consumers and 600 providers. Single scaling refers to the scaling of number of iterations with $s_f^{1.75}$, and double scaling refers to the scaling of iterations and number of particles/individuals times s_f.

The evaluation showed that for larger problems, i.e., larger consumer-provider pairs, at least the iterations have to be scaled in order to keep the results of the DGC-PSO and GA-E algorithms on the same high match scores. If a fast service selection time is required for larger problems, only the number of iterations could be increased, yet the scaling factor has to be raised to a higher power. If the factor s_f is not raised to a higher power for this case, the match score will decrease for a higher number of consumer-provider pairs. In this particular case, DGC-PSO showed better results than the GA-E. This further suggests, that if the number of iterations and number of particles/individuals are scaled with an even higher factor than s_f, the two methods will probably be able to find the optimal result. However, the execution time of the algorithms will increase drastically.

As a recommendation, if the match quality of consumers and providers is paramount, then the Munkres algorithm should be chosen, taking into consideration that the execution time can be very high for large numbers of consumers and providers. If on the other hand speed is paramount, then both the DGC-PSO and the GA-E approaches should be chosen, preferably the DGC-PSO approach as it achieves slightly higher overall match scores. For small number of consumer-provider pairs the GA-E is slightly faster, for large number of consumer-provider pairs the DGC-PSO is faster, i.e., the scalability is better.

Future work will implement a non-dominated sorting GA, and a multi-objective PSO approach in order to compare it with the current approaches. Also, different selection matching functions will be tested.

REFERENCES

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

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

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

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

[5] M.C. Jaeger, G. Mühl, 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, 2007.

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

[7] J. Kennedy and R. Eberhart, Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks*, 1995.

[8] M. Clerc, Discrete particle swarm optimization - illustrated by the traveling salesman problem, New Optimization Techniques in Engineering, Springer, 2004.

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

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

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

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

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

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