2011 IEEE Asia -Pacific Services Computing Conference

A Tabu Search Approach for Dynamic Service Substitution in SOA Applications

Can Zhang
School of Software
Shanghai Jiao Tong University
Shanghai, China
Email: zhangcan@sjtu.edu.cn

Haopeng Chen
School of Software
Shanghai Jiao Tong University
Shanghai, China
Email: chen-hp@sjtu.edu.cn

Jinbo Du
School of Software
Shanghai Jiao Tong University
Shanghai, China
Email: dujinbo@gmail.com

***Abstract*—Due to the possibility of internal changes of services or changes in their environment, dynamic service substitution has become a key problem in Service Oriented Architecture (SOA). In recent years, researchers have worked out several algorithms for dynamic service substitution. However, the performance of these algorithms is generally not that satisfactory. In this paper, we propose a heuristic service substitution algorithm based on Tabu Search. The algorithm models the service substitution problem as searching valid solutions in the Candidate Service Graph (CSG). In the end, the effectiveness and performance of the algorithm are studied by simulations. The simulation results show that the algorithm performs very well in dynamic adaption, especially when the scale of the service substitution problem surges.**

***Keywords*—tabu search; service substitution; SOA;**

I. Introduction

In Service-Oriented Architecture (SOA), several services from different service providers are composed together to fulfill complex business needs. All the component services in a SOA application work as a whole to service client requests. However, during the execution of a SOA application, if service failure is detected, a service substitution mechanism is needed to ensure that the failed service should be replaced effectively and efficiently. In fact, the problem of service substitution covers several aspects, including how to monitor the runtime status of the SOA applications, how to identify those services which have same functions but different non-functional properties, how to find the replacement solution efficiently, and how to reconfigure the service workflow without interrupting the running processes. Our previous work has presented feasible solutions to some of these problems. In [1], we use feedback mechanism to capture the runtime QoS attributes of each component service, and provide three methods to collect the real-time QoS data [2]. With these dynamic data, a novel and extensible ontology-based approach [3] can be applied to describe the QoS constraints and calculate the QoS attributes of the composite service. Also, a P2P service registry called QMC is designed and implemented for storing QoS feedbacks, managing QoS data, and handling QoS-based service discovery requests [4]. What's more, in [5], Fredj et al. proposed the SIROCO middleware platform, which enables the runtime, semantic-based service substitution.

In this paper, we mainly focus on the algorithmic aspect of the service substitution problem. One of the most challenging issues in service substitution is to manage the time complexity of the algorithm. Because of their similarity, service substitution is closely related to the service selection problem. Moreover, the service selection problem can be mapped to a multidimension multichoice 0-1 knapsack problem (MMKP) [6], which has been shown to be NP-complete [7]. On the other hand, heuristic algorithms are used to speed up the process of finding a satisfactory solution, where an exhaustive search is impractical. Consequently, heuristic algorithms can be introduced to solve the service substitution problem, and Tabu Search is one of the well-known local search algorithms [8]. What's more, the way how Tabu Search works makes it a suitable algorithm for service substitution. For a composite service which may be composed of a large number of services, upon failure time, generally only a small number of services shall be replaced. At the same time, a typical Tabu Search algorithm starts from a randomly-generated initial solution and iteratively searches in its neighborhood until it finds a locally-optimal solution. As a result, if we simply use the old composite service as the initial solution, it is possible that after a few iterations, the algorithm will reach a valid solution. Even in the worst-case scenarios, the algorithm may travel through the whole solution space, which is as good as, if not better than the brute force method.

The rest of the paper is organized as follows. Section 2 surveys some related work. Section 3 introduces a motivating example to explain the problem discussed in this paper. Section 4 presents the problem model defined for the service substitution algorithm and the notations used in the paper. Section 5 shows the Tabu Search-based algorithm for dynamic service substitution. The simulation results of performance study are shown in Section 6. Finally, the paper is concluded in Section 7.

II. Related Work

Many researchers have worked on the dynamic service substitution problem. A native approach to handle the problem is to re-compose from scratch every time a change occurs. In [9], Liangzhao Zeng et al. models the service composition as a linear programming problem, and provides an efficient algorithm to dynamically construct the composite service according to qualitative criteria like price, duration, reliability etc. However, this may not be a feasible solution for its high time complexity.

In [10], Tao Yu et al. divide the service substitution problem into two situations. First, for the business processes that have executed half way, they will be switched to a backup path that bypasses the failed service node. Second, for new business process instances, they will be re-configured to use a new globally optimal path without using failed service node. All the locally- and globally- optimal paths are stored by a broker, so the system can efficiently switch to a backup process when failure happens. However, the approach presented in [10] can only handle a single point of failure. Also, we cannot ensure the availability of the backing up processes when failure happens.

Part of our problem modeling is motivated by the business process model presented in [10], which builds a directed acyclic graph (DAG) based on the abstract model using all Web service candidates. The main difference is that in [10], every edge has delay, cost, and benefit weights, while every node is associated with a non-functional vector in our model.

Saboohi et al. provide a subgraph replacement approach [11] to recover from a composite service failure. Upon failure, a service broker is invoked, trying to re-execute the failed constituent web services. If re-execution attempts fail, a failure recovery strategy by replacing a sequence of semantic web services is applied. By searching for replacement alternatives of all the subgraphs which are compatible to the original one, the best ranked alternative subgraph will be the replacement selection. The limitation is that the time complexity of calculation of all subgraphs is too high, for the number of all subgraphs is $(n(n+1))/2$, where n is the number of component services.

In [12], Moser et al. present a non-intrusive monitoring and service adaption system called VieDAME. It can monitor BPEL processes, discover QoS failure, and replace failed partner services based on some replacement strategies. In [12], Moser et al. highlights VieDAME's ability in monitoring BPEL processes and its role as SOAP message mediator when interface mismatches occur in replacement. However, they pay little attention to the complexity of the replacement algorithms, which may become the performance bottlenecks of the whole system.

In summary, to dynamically replace the failed composite service, we have to provide an efficient and effective algorithm so that the availability of the candidate services is ensured and also the service process is not interrupted.

III. A Motivating Example

To better illustrate the problem discussed in this paper, we consider an online travel agency which is composed of five services: login service, flight booking service, attraction ticket booking service, hotel booking service, and logout service (see Fig. 1). Suppose that the online travel agency will automatically finish the three booking tasks after all the information needed is input by the clients.

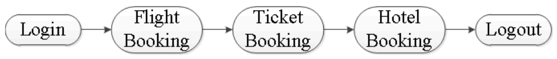

Fig. 1. The Motivating Example of an Online Travel Agency

If all the services respond quickly, then the clients will finish their requests in no time (the time need to complete the whole booking service is the sum of the time of flight booking service, the attraction ticket booking service and the hotel booking service). However, if one of the services experiences a large surge of clients, maybe it will respond much slower than expected or do not respond during to interval failure. For that matter, a service substitution is needed. For a small composite service composed of several services (here, the online travel agency), it may be easy to detect the failed service and substitute it with a good one. But for a composite service composed of hundreds of services, it may not be that easy to detect the failed service. What's worse, the slow response of the whole system may not be caused by a single failed service. Rather, several services in the system become slower than before. Then, which services in the system should be replaced to make the composite service satisfy the QoS requirements again? Especially when the number of candidate services on the Internet is huge, it takes some time to find a substitution solution for current composite services.

IV. Problem Model

A. Notations

The following notations are used in this paper:

1) *s*: a single service instance running on an addressable server on the Internet.
2) *S*: a set of services which have the same functionality but different non-functional properties.
3) *cs*: a composite service. A composite service is a complex service process composed of a set of services which can together fulfill complicated functions. In a composite service, if the output data of service s_1 are the input data of service s_2, then we say service s_1 is the predecessor of s_2, or s_2 is the successor of s_1. Also, we shall use the notation $s_1 \to s_2$ to represent that s_1 is the predecessor of s_2 (or s_2 is the successor of s_1). In this paper, we only discuss the sequential composite service, which means each service s (except the first node and the last one) in the composite service has a single predecessor and a single successor (the first node only has a single successor and the last one only has a single predecessor). For those parallel, conditional or loop systems, we can use the reduction function in [3] to reduce them to sequential systems.
4) *q(s)*: non-functional property of service s.
Defintion 1 For two services s and s', if $q(s)$ is better than $q(s')$, then we say that $q(s) > q(s')$.
5) *V(q(s))*: the value of the non-functional property $q(s)$. For some properties, like the availability, reliability, the bigger $V(q(s))$ is, the better the quality $q(s)$ is. We call

these properties *positive properties*, otherwise *negative properties*.

6) *Q(s)*: the vector of non-functional properties, denoted by

$$Q(s) =< q_1(s), q_2(s), \cdots, q_n(s) >$$

For example, if the non-functional properties include the price, response time, availability and reliability, then

$$Q(s) =< q_{price}(s), q_{rt}(s), q_{re}(s) >$$

We use Definition 2 to compare two quality vectors. For example, let $Q_1(s)$ be the non-functional requirements, then we say $Q_2(s) < Q_1(s)$, if there's any property in $Q_2(s)$ that cannot satisfy the requirement in $Q_1(s)$.

Definition 2 For a quality vector

$$Q(s) =< q_1(s), q_2(s), , q_n(s) >$$

we say that $Q(s_1) \geq Q(s_2)$, if

$$q_m(s_1) \geq q_m(s_2), \forall m \in [1, n]$$

Similarly, $Q(s_1) < Q(s_2)$, if

$$\exists m \in [1, n], q_m(s_1) < q_m(s_2)$$

B. Problem Description

Formally, we define the service substitution problem as below:

Definition 3 let cs_{old} be the old composite service, and cs_{new} be the new composite service. Also, the non-functional requirements vector for the composite service is

$$Q(cs_{req}) =< q_1^{req}, q_2^{req}, q_3^{req}, \cdots, q_n^{req} >$$

The service substitution problem is:
At the time when $Q(cs_{old}) < Q(cs_{req})$, to find a new composite service cs_{new} such that $Q(cs_{new}) \geq Q(cs_{req})$ holds.

V. The Tabu Search Solution

A. Candidate Service Graph (CSG)

The solution space of our service substitution algorithm is the *Candidate Service Graph*.

Definition 4 A Candidate Service Graph is a DAG that satisfies three requirements:

1) If service s_j is a candidate service for service s_i, then s_j will inherit all the predecessors and successors from s_i. In other words, the predecessors and successors of s_i will be the predecessors and successors of s_j.
2) If service s_p is the predecessor or successor of service s_q, then all the candidates for service s_p will be the predecessors or successors of s_q.
3) A virtual source node(v_s) and a destination node(v_d) are added to the graph. Also, add an edge from v_s to those who do not have income, and an edge from those who do not have outcome to v_d.

Fig.2 illustrates a simple Candidate Service Graph. In Fig.2, services s_{12},s_{13} are the candidates for service s_1, services s_{22},s_{23} candidates for service s_2, services s_{32}, s_{33} candidates for service s_3, and service s_{42} candidate for service s_4.Thus the Candidate Service Graph for the sequential composite service $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$ is shown in Fig.2.

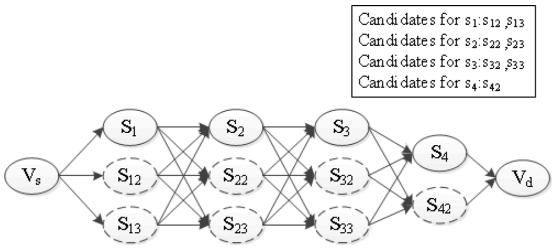
Candidates for s1:s12 ,s13
Candidates for s2:s22 ,s23
Candidates for s3:s32 ,s33
Candidates for s4:s42
Vs
S1
S12
S13
S2
S22
S23
S3
S32
S33
S4
S42
Vd

Fig. 2. Candidate Service Graph

In the following section, we will describe our service substitution solution organized by the basic components in Tabu Search algorithm, including objective function, move, neighborhood, candidate list, etc.

B. Objective Function

Let the quality requirements vector be

$$Q(cs_{req}) =< q_1(cs), q_2(cs), q_3(cs), , q_n(cs) >$$

we give each non-functional property an evaluation ratio, and for each non-functional property q_j, we use the following formula to calculate its evaluation ratio u_j:

$$u_j = \begin{cases} \dfrac{V(q_j^{req}) - V(q_j)}{V(q_j^{req})}, & \text{where } q_j \text{ is negative (1a)} \\ \dfrac{V(q_j) - V(q_j^{req})}{V(q_j^{req})}, & \text{where } q_j \text{ is positive (1b)} \end{cases}$$

where $V(q_j^{req})$ is the quality requirement for a given non-functional property, and $V(q_j)$ is the actual value of that property.

The equation below is the evaluation function for a composite service:

$$Score(cs) = \sum_{j=1}^{n}(w_j * u_j) \tag{2}$$

where $w_j \in [0, 1]$, and $\sum_{j=1}^{n} w_j = 1$. Here, $w_j(j = 1 \ldots n)$ is the weight or impact factor of each property, and its value can be set by the programmer manually or by the analysis of the user preference. If the programmer considers the availability to be an important quality property, then he/she can give a significantly-larger weight for the coefficient w_{av} before the *evaluation ratio* of availability. Also, the values of $w_j(j = 1 \ldots n)$ can be obtained by analyzing user preference data through machine learning methods, like genetic algorithms or neural networks. If the values are not set, the default value of $w_j(j = 1 \ldots n)$ is $1/n$.

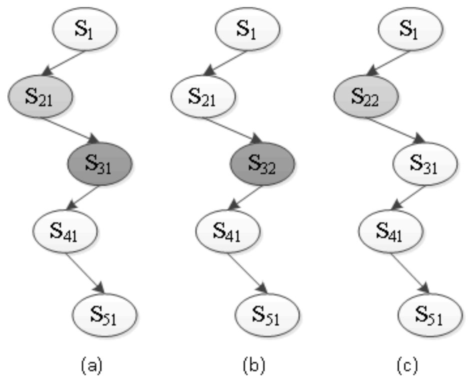

Fig. 3. A Move

C. Move

Definition 5 For a composite service, to replace a single component service with its candidate is a *move*.

Consequently, the least number of moves needed from one composite service to another is the distance between the two composite services.

In Fig.3, services s_{21},s_{22} belong to the same service set S_1, and services s_{31},s_{32} belong to another service set S_2. Thus, if we replace service s_{31} with s_{32} in the composite service (a), then the composite service (a) is transformed to (b). So, we say that the distance between (a) and (b) is one move. Similarly, the distance is also one move between (b) and (c), and two moves between (a) and (c).

D. Neighborhood

Definition 6 Two composite services are neighbors when the distance between them is exactly one move.

According to Definition 6, in Fig.3, composite services (a) and (b), and composite services (b) and (c) are neighbors, while (a) and (c) are not.

Definition 7 Given a composite service cs, the neighborhood $N(cs)$ is a set of all neighbors of cs.

Therefore, in Fig.3, composite services (a) and (c) are in $N(b)$.

E. Candidate List

Suppose that the given composite service cs is composed of hundreds of services, and each service has about one hundred candidates, then the neighborhood $N(cs)$ may become a huge set. It is inefficient to search the whole neighborhood. Therefore, we have to select some good neighbors from its neighborhood and add them to the candidate list. In our service substitution algorithm, the candidate list consists of those neighbors who maximize the value of the objective function.

F. Tabu List and Tabu Tenure

In our algorithm, the Tabu tenure is defined to be a constant. Also, we design two Tabu lists, namely *Tabu-active* list and *Tabu-inactive* list. Tabu-active list consists of those services which have just replaced an old service in the composite service, while Tabu-inactive list is composed of those services which have just been replaced. The difference between the two Tabu lists is that services in the Tabu-active list cannot be dropped during the Tabu tenure, while the ones in the Tabu-inactive list cannot be selected during the Tabu tenure. However, there's one exceptional condition: if the new composite service satisfies the aspiration criteria defined in section G, the move can be made even if the involved services stay in either of the two lists.

G. Aspiration Criteria

In our service substitution algorithm, if and only if the solution is the best solution (the value of the objective function is maximum) that has been found so far, the Tabu status of the service can be ignorable.

H. Long-term Memory

We record the times that each component service has been added to the Tabu lists (Tabu-active list and Tabu-inactive list). Once the optimal solution hasn't been improved for a given number of iterations, we will restart the search from a new composite service composed of those least-visited services.

I. Terminating Conditions

When the composite service that satisfies quality requirements is found or the maximum number of iterations has been reached, the algorithm terminates.

J. Service Substitution Algorithm

In this section, pseudo code is used to describe the basic ideas of our service substitution algorithm.

1) Definitions of the identifiers
   - cs: the old composite service
   - qos: the non-functional requirements vector
   - $tabuActiveList$: the Tabu-active list
   - $tabuActiveTenure$: the Tabu-active tenure (a constant)
   - $tabuInactiveList$: the Tabu-inactive list
   - $tabuInactiveTenure$: the Tabu-inactive tenure (a constant)
   - s: current composite service
   - $s*$: the optimal solution found so far
   - $f*$: the maximum value of the objective function found so far
   - $numOfIterations$: current number of iterations
   - MAX: the maximum number of iterations
   - $MAX\_NOIMPROV$: the maximum number of iterations that the optimal solution can stay unimproved
   - $< old\_s, new\_s >$: a move that the old service old_s is replaced with new_s
2) Pseudo code

```
Algorithm 1: DoReplacement
input : The old composite service $cs$
input : The requirements vector $qos$
1  constructCandidatesGraph($cs$) ;
2  $tabuActiveList$ = [] ;
3  $tabuActiveTenure$ = 1 ;
4  $tabuInactiveList$ = [] ;
5  $tabuInactiveTenure$ = 2 ;
6  $s = cs$ ;
7  $s^* = s$ ;
8  $f^* = f(s^*)$ ;
9  numberOfIterations = 1 ;
10 while numberOfIterations $< MAX$ do
11     choose a best neighbor
       $s' = s+ < old\_s, new\_s >$
       such that $old\_s \notin tabuActiveList$
       and ($new\_s \notin tabuInactiveList$ or $f(s') > f^*$) ;
12     $s = s'$ ;
13     addServiceToTabuActiveList($new\_s$) ;
14     addServiceToTabuInactiveList($old\_s$) ;
15     update tabu lists ;
16     if $f(s) > f^*$ then
17         $s^* = s$ ;
18         $f^* = f$ ;
19     end
20     if satisfy(qos,s) then
21         Break ;
22     end
23     if noImprovement($f^*$,$MAX\_NOIMPROV$) then
24         $s$ = constructLeastVisitedSolution() ;
25     end
26 end
```

VI. Simulation Experiments

For a composite service composed of m services, and $services_i (i = 1 \dots m)$ has b_i candidate services, then the size of the solution space is $n = \prod_{i=1}^{m} (b_i + 1)$.

To test the efficiency of the Tabu search solution, we have conducted several experiments, whose solution space ranges from 10^2 to 10^{11}. In the following sections, we will present an experiment which has a small solution space, and the results of experiments with solution spaces from 10^2 to 10^{11}.

A. Experiment No.1

In this experiment, we construct a sequential composite service which is composed of four abstract services, and each service has two to three candidates. Also, the QoS properties include the availability, the price, the reliability, and the response time.

TABLE I
Non-functional Properties of Services

Service	Availability	Price	Reliability	Response Time
s_1	0.98	300	0.90	15
s_{11}	0.98	400	0.95	10
s_{12}	0.96	200	0.90	30
s_{13}	0.97	300	0.90	20
s_2	0.95	500	0.94	60
s_{21}	0.98	600	0.95	50
s_{22}	0.95	500	0.95	40
s_{23}	0.98	400	0.95	45
s_3	0.97	500	0.90	30
s_{31}	0.97	400	0.91	25
s_{32}	0.96	350	0.93	35
s_{33}	0.97	400	0.92	30
s_4	0.98	800	0.90	20
s_{41}	0.98	700	0.90	30
s_{42}	0.97	750	0.90	10

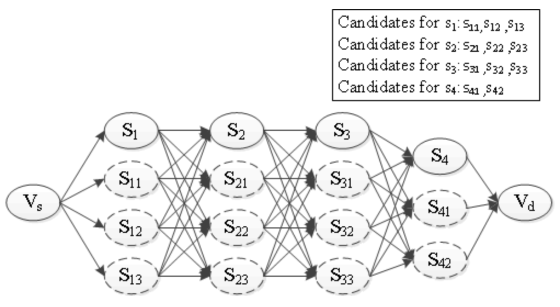
Candidates for s1: s11, s12, s13
Candidates for s2: s21, s22, s23
Candidates for s3: s31, s32, s33
Candidates for s4: s41, s42
Vs
S1
S11
S12
S13
S2
S21
S22
S23
S3
S31
S32
S33
S4
S41
S42
Vd

Fig. 4. CSG for Experiment No.1

The non-functional requirements for the system is

$$Q(cs_{req}) =< availability, price, reliability, responsetime > =< 0.90, 2000, 0.60, 100 >$$

Table I lists the non-functional properties of all the services, including component services in the old composite service and all the candidates. Also, Fig.4 shows the CSG for the service substitution problem.

After calculation, we can see that the old composite service cannot satisfy the non-functional requirements, and the service substitution algorithm is invoked.

B. Experiment No.2

In the second experiment, we randomly generate ten test cases, where the solution space varies from 10^2 to 10^{11}. Table II shows the simulation results: where n_{actual} is the actual solution space visited in the algorithm, n_{space} is the whole solution space, and $p = \frac{n_{actual}}{n_{space}}$.

From the results, we can see that our service substitution algorithm can find the optimal result by searching a small portion of the whole solution space, especially when the size of the solution space gets larger.

VII. Conclusion and Future Work

In this paper, we presented an effective heuristic service substitution algorithm. First, we model the service substitution

TABLE II
SIMULATION RESULTS

Experiment	n_{actual}	n_{space}	**p**
No.1	20	10^2	0.2
No.2	36	10^3	3.6×10^{-2}
No.3	68	10^4	6.8×10^{-3}
No.4	150	10^5	1.5×10^{-3}
No.5	2676	10^6	2.7×10^{-3}
No.6	3156	10^7	3.1×10^{-4}
No.7	6952	10^8	7.0×10^{-5}
No.8	18702	10^9	1.9×10^{-5}
No.9	24890	10^10	2.5×10^{-6}
No.10	31911	10^11	3.2×10^{-7}

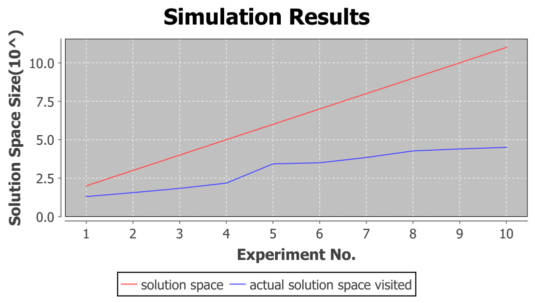

Fig. 5. Simulation Results Diagram

problem as searching valid solutions in the Candidate Service Graph. Then, we solve it with a heuristic algorithm which is based on the ideas of Tabu Search. The simulation results show that our algorithm performs well, especially when the problem size gets larger.

Nevertheless, more work has to be done for tuning the parameters in the algorithm, including the Tabu tenures for the active and inactive Tabu lists, and the size of the candidate list. Currently, the Tabu tenures are set to be constants, and the candidate list only contains those solutions that maximum the value of the objective function. In order to improve the performance of the algorithm, more experiments are needed to decide whether the Tabu tenure should be a constant or the candidate list should also include candidates with other features.

REFERENCES

[1] H. peng Chen, G. Yang, and C. Zhang, "A closed-loop mechanism for service evaluating and discovering on the internet," in *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, aug. 2009, pp. 1 –8.

[2] C. Zhou and H. Chen, "A mechanism for collecting and feedbacking the real-time quality of web service," in *Information Science and Engineering (ICISE), 2009 1st International Conference on*, dec. 2009, pp. 2802 –2807.

[3] M. Li, H. peng Chen, and N. Wang, "The description and calculation of quality of composite services 1," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, dec. 2009, pp. 385 –390.

[4] S. Xiong and H. Chen, "Qmc: A service registry extension providing qos support," in *New Trends in Information and Service Science, 2009. NISS '09. International Conference on*, 30 2009-july 2 2009, pp. 145 –151.

[5] M. Fredj, N. Georgantas, V. Issarny, and A. Zarras, "Dynamic service substitution in service-oriented architectures," in *Services - Part I, 2008. IEEE Congress on*, july 2008, pp. 101 –104.

[6] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Trans. Web*, vol. 1, May 2007. [Online]. Available: http://doi.acm.org/10.1145/1232722.1232728

[7] M. Fischetti, S. Martello, and P. Toth, "The fixed job schedule problem with spread-time constraints," *Operations Research*, vol. 35, pp. 849–858, 1987.

[8] E. H. L. Aarts and J. K. Lenstra, "Local search in combinatorial optimization," 1997.

[9] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *World Wide Web Conference Series*, 2003, pp. 411–421.

[10] T. Yu and K.-J. Lin, "Adaptive algorithms for finding replacement services in autonomic distributed business processes," in *Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings*, april 2005, pp. 427 – 434.

[11] H. Saboohi, A. Amini, and H. Abolhassani, "Failure recovery of composite semantic web services using subgraph replacement," in *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on*, may 2008, pp. 489 –493.

[12] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for ws-bpel," in *World Wide Web Conference Series*, 2008, pp. 815–824.