Dynamic Asynchronous Aggregate Search for solving QoS compositions of Web services

Xuan Thang Nguyen, Ryszard Kowalczyk, and Khoi Anh Phan
Faculty of Information and
Communication Technologies
Swinburne University of Technology
Melbourne VIC 3122, Australia
Email: {xnguyen,rkowalczyk}@ict.swin.edu.au

***Abstract*— There has been recently an increase of interests in Distributed Constraint Satisfaction (DisCSP) algorithms. Most of these algorithms assume that the set of variables and constraints in a DisCSP are completely known and fixed. However, these assumptions do not hold in many open environments where unexpected events and changes can happen. Therefore this issue needs to be addressed in order for a DisCSP algorithm to be practically used in real life applications. Our contribution in this paper is two-fold. Firstly, we present an extended version of the Asynchronous Aggregate Search (AAS), called the Dynamic Asynchronous Aggregate Search (DynAAS) - as a new algorithm for dynamic and uncertain environments. Secondly, we investigate the applicability of DynAAS in the Web service domain and argue that DynAAS is suitable for solving the QoS compositions of Web services. We also carry out the experiments to show the efficiency of our proposed algorithm in solving the QoS composition problem.**

I. Introduction

Constraint Satisfaction has been considered as a powerful paradigm for generic problem solving and found in many real life applications ranging from resource allocation and scheduling to system configuration and design. The success of CSP is based largely on the efficiency of its algorithms. A distributed CSP (DisCSP) is a CSP when variables and constraints are distributed among different solving entities. Over the past few years, there has been an increasing interest in the agent community for developing different DisCSP algorithms and techniques. Unfortunately, little research has been done on solving dynamic DisCSPs in which the DisCSPs may change over time.

There are many situations in which the changing of a CSP are desirable. This is especially true for problems defined in environments with unpredictable events and changes. For example, online planning and scheduling, computer-aided system design or configuration are problems where uncertainties cannot be easily avoided. The notion of dynamic CSP [10] has been introduced to represent such situations where constraints can be added or removed. It is easy to see that without loss of generality, this notion cover all possible changes to a CSP (domain modifications variable additions or removals can be modeled as constraints.)

This paper presents an extended version of AAS algorithm, a selected DisCSP, for dynamic environment. The paper is organized as follows. In the next section, we describe the *QoS guarantees for multiple Web service compositions*- finding a solution for this problem is our motivation in developing DynAAS. An overview of AAS algorithm is presented in Section III. In Section IV, we discuss the AAS extension as DynAAS. We present our experiments to show the efficiency of DynAAS performance in Section V. Finally, conclusions and future work are discussed in Section VI.

II. Motivation

During the past few years, in an effort to improve the collaborations between organizations, the Web service framework has been emerging as a de-facto choice for integrating distributed and heterogeneous applications across organizational boundaries. Consequently, much research has been carried out in various areas including Web service discovery, composition, and management. Web service composition in general focuses on building a new value-added *composite* Web service from a number of existing *component* Web services. For a composite service, in addition to its functional requirements, QoS requirements are important and deserve a special attention. The central question to a QoS composition problem is how to compose a service from different sub-component services so that its overall QoS can satisfy certain requirements. Solving the QoS composition problem is the motivation that leads us to investigate the applicability of DisCSP techniques. In this part, we argue that in general DisCSP algorithms are suitable to solve this problem.

Since a service provider must allocate necessary resources to live up to the QoS guarantees, if its service engages in a number of compositions, there will be a dependency between the levels of QoS that service can contribute to these compositions. Consequently, there is a mutual relationship between the compositions. Here we assume that this relationship can be formally expressed as constraints. These constraints might be shaped by the provider's resource limitations, business rules, organizational policies or even conditions in contracts with a third party. The providers have a choice to reveal them or not by making the constraints *shared* (i.e. known to a number of or all other providers) or *private* respectively. Here we focus on a general problem in which multiple providers engage in multi-Web service compositions. The final goal of the QoS

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE

guarantee for multiple Web service compositions is to satisfy the E2E QoS requirements of all compositions.

It is important to note that our problem of QoS guarantees for multi-Web service compositions has a different focus as compared to other work on QoS compositions, e.g. [4], [3], [1]. We focus on multiple related compositions which are composed by multiple providers whereas those work focus on single composition composed by a single *QoS broker*.To apply DisCSP techniques for solving the problem of QoS guarantee for multi-Web services compositions, each service provider can be considered as an agent (an autonomously processing entity) in a constraint network. Each QoS parameter is mapped into a variable in the constraint network; the set of providers' constraints is mapped into the network's constraint set. For the rest of this paper, we will use the terms *service providers* and *agents* interchangeably. More formally, the problem of QoS guarantee for Web service compositions, if in a static environment, can be considered as an instance of DisCSP problems.

To take into account the dynamic nature of Web services environment, in the DisCSP framework, the appearance of a new composition indicates that new constraints and possiblly new variables and agents are added into the constraint network. Dissolving of a composition means that some existing constraints are removed and possibly some existing variables or agents are also removed. In general, there are introductions or reductions of new variables, constraints, and agents. Hence, this is a Dynamic DisCSP.

III. Asynchronous Aggregate Search

For the application of DisCSP techniques in the Web service environment each agent needs to handle many QoS variables and complex local constraints. The originial DisCSP model [12] and most of the solving algorithms , as discussed before, focus on shared constraints instead of private constraints and hence is more suitable for distributed control but not negotiation [7]. An exception is Asynchronous Aggregate Search (AAS) [7] that allows one agent to maintain a set of variables and these variables can be shared and hence is suitable for negotiation. All constraints are private in AAS, however shared constraints can be modeled as duplicated private constraints. AAS is selected for solving the QoS guarantee problem due to these reasons.

AAS [7], [9] is a DisCSP search technique based on the classical ABT (Asynchronous Backtrack) algorithm [12]. In AAS, each agent maintains a set of variables which can be shared with others and a set of intra-constraints. It differs from most of existing methods in that it exchanges aggregated consistent values (in contrast to a single value in ABT) of partial solutions. For example, in ABT, only a single value (*single assignment*) is sent. The aggregated consistent values are the Cartesian products of domains which represent a set of possible valuations. This technique significantly reduces the number of backtracks.

IV. Dynamic Asynchronous Aggregate Search

A. AAS and local solvers

Algorithm 1: CSP solvers for local solving

```
1  procedure check_agent_view do
2      when agent_view and current_inst_aggregate are not
       consistent do
3          S = localsover.solve(agent view, local
           constraints);
4          if S=∅ then
5              backtrack;
6          else
7              boxes_partition(S);
8              construct and send out ok? messages to A_k^-;
9  procedure B=boxes_partition(S) do
10     B=∅;
11     while S≠∅ do
12         S_box = select any point∈S;
13         foreach dimension d do
14             extend S_box along d;
15         B= B∪S_box;
16         S=S\{s : s ∈ S_box};
17     return B;
```

Existing search algorithms like ABT assume that either each agent holds only one variable; or its private constraints are simple. As such, there has been not much work to solve the DisCSP problems with complex local constraints. However in AAS, each agent normally has many variables and every constraint is private. In addition, due to dynamic property, the set of constraints are not the same over time. Consequently, the complexity of finding local solutions can no longer be neglected. To take this complexity into the account and make AAS more flexible for implementations, we propose to employ a CSP solver inside each agent to solve its own private constraints. In this paper, we suggest that CSP solvers can be considered as black boxes. The local CSP solver implementation is private to each agent. In this way, proven CSP algorithm and techniques can take part in the overall DisCSP solving process. For an agent A_k, we denote A_k^+ the set of linked agents whose priorities are higher, and A_k^- the set of linked agents whose priorities are lower than the current agent's priority. V^+ is the set of variables the agent share with A^+, and V^- is with A^-. Everytime an agent needs to find out a consistent assignment (with its current view), it invokes its local CSP solver. The CSP solver of agent A takes assigments from A_k^+ and generates solutions for V_k^-. We call this process a *local solving process*. In addition to A_k's local constraints, the nogoods recorded by A_k are also the CSP solver's input constraints for a local solving process. Whilst incorporating a CSP solver into each AAS agent allows the AAS algorithms to be more flexible as many successful constraint solving tools are available today such as ILOG JSolver [5], and NSolver [6]. In AAS, for effiency of nogood storage and processing,

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE

assignments are normally in the forms of Cartesian products of value ranges (consecutive values in a domain), i.e. boxes. This can done by employing existing methods like the ones proposed in [2], [8] which maintain outputs of boxes for *local solving process*. However, there is no guarantee that the outputs from a constraint solving tool is always boxes. We address this by introducing a greedy algorithm (boxes_partition in Algorithm 3) to construct boxes from a set of points (i.e. solutions in a vector space). This algorithm picks up a point in the solution set and systematically extends the point as a box along each dimension (i.e. variable domain) as long as all points contained in this box are solutions (line 12 to 14). All solutions in the box's volume are removed from the solution set and the whole process is started again util the solution set become empty. boxes_partition algorithm returns the list of distinct boxes which cover the solution set.

B. DynAAS Algorithm

Our new extension of AAS for dynamic environment is based on an indexing technique called *eliminating explanation* which had been proposed in centralized DynCSP [10]. Note that (nogood based) CSP algorithms in general generate and test solutions, and record nogoods (invalid solutions). The main idea of the *eliminating explanation* technique is simple enough: to index every nogood against the minimal set of constraints that create the nogood, and remove the nogood if a constraint in the constraint set is removed. In DynAAS, an agent creates and stores an *eliminating explanation* before it sends a nogood. The sent nogood is tagged with an identity number and kept by both the sender and the receiver so that the sender, due to some changes later, can ask the receiver to remove this nogood. It does this by sending the receiver a *remove-nogood* message that contains the nogood's identity.

The procedure *add-constraints* shows the processing carried out at an agent to handle a newly added constraint set C_{new}. In the algorithm, the agent first tries a local repair of the partial assignments of variables contained in both those constraints (i.e. v(C_{new})) and V_k^+ (line 2). If it fails, the agent then attempts to repair the assignments of the whole V_k^+ (line 4). New assignments if exist are used to update the view and sent to A_k^+, otherwise a backtrack occurs.

The procedure *remove-constraints* is used to the removal of a constraint set $C_{removed}$. The agent bases on its eliminating explanation set (E) to detect which nogood it sent to a parent in the past is no longer a nogood (line 2). It then sends a *remove-nogood* message to ask the parent for the removal of this nogood. The nogood is a constraint from the parent's perspective. Therefore, the parent handles the message by invoking Algorithm 2. Adding and removing constraints are the same as calling *add-constraints* and *remove-constraints* sequentially.

The DynAAS algorithm processes *ok?* and *nogood* incoming messages in the same way as AAS (Algorithm 1 in the previous section). When a remove-nogood message is received, the agent A_k views the nogood in the message as one of its constraints hence it invokes the procedure *remove-constraint* to remove this nogood from the nogood list. This action may further lead to sending *remove-constraint* messages to its higher priority neighbors. For an example in Figure 2, the agent A_3 after removing **C**, looks up its own eliminating explanation list to identify the corresponding nogoods' identities that it sent in the past. A_3 then sends these nogoods' identities to A_2. A_2 receives the message and translate this as a request of dynamic removal of all local constraints that have the identities in the message. These local constraints may include the nogoods previously received from A_3. A_2 again repeats the process carried out by A_3.

Algorithm 2: DynAAS remove-constraint messages

```
1  when received remove-constraint do
2      identify the nogoods N_child defined in the
       remove-constraint message;
3      remove_constraints(N_child);
4      check_agent_view;
```

Algorithm 3: Adding and Removing constraints

```
1  procedure add_constraints(C_new) do
2      update neighbor list, variable list, and constraint list;
3      assgmts = re-assign(v(C_new) ∩ V_k^+) ;
4      if assgmts = ∅ then
5          assigmts =re-assign(V_k^+);
6      if assgmts = ∅ then
7          send nogoods to A_k^-;
8      else
9          update view and send ok to A_k^+;
10 procedure remove_constraints(C_removed) do
11     update neighbor list, variable list, and constraint list;
12     forall e ∈ E and c(e) ∩ C_removed ≠ ∅ do
13         send remove-nogood message to parent to remove
           the nogoods c;
14         delete e from E;
```

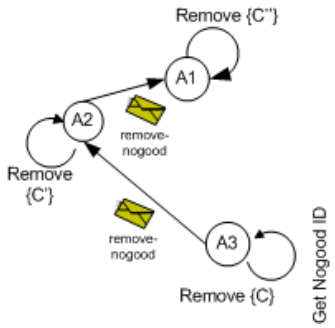

Fig. 1. Constraint removal example

It is important to note that adding or removing of variables or domain values can be modeled as constraints [11], therefore

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE

can be handled by the two above algorithms. If a new agent is added to the network, it is given the lowest priority. For every type of changes, affected agent must update its lists of neighbors, variables, and constraints (e.g. line 1 of Algorithm 1 and 2) first.

As we have seen so far, the key idea of DynAAS is to reuse partial solutions to achieve stability. If we model the whole environment as a discrete-event system where events are constraint additions or removals, then during the interval between any two consecutive events the system can be viewed as a static DisCSP using AAS. This is because DynAAS reacts to maintain consistent views and nogood storages whenever constraints are added or removed.

V. Framework and Experiments

In our experiment, 10 Web services are used. A list of prefetched constraints on QoS parameters, and neighbors (i.e addresses of other DisCSP Web services) are stored in a mySQL database. These data are modified by a simulator with the varying rate σ of adding/removing constraints. In particular, compositions are randomly formed and removed among any 3 agents. For the sake of clarity, each of our compositions consists of exactly three component services and introduces maximum two constraints on the E2E QoS parameters of the composition. The QoS parameters that we use are response time and cost which both have simple aggregation formulas [3]. Each parameter has a domain of 10 discrete values. The average of all constraint tightness (the probability that there is not a valid assignment within a constraint) is 30% . 20 instances are run for each test.

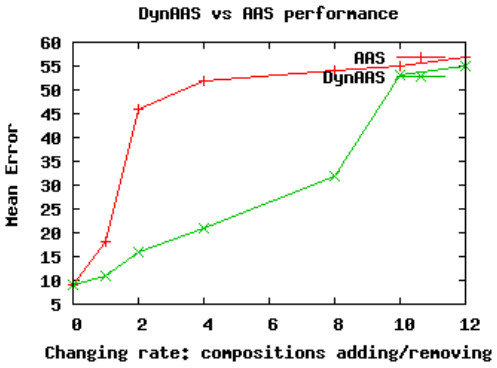

Fig. 2. Mean Error of DynAAS versus Static AAS for 10 providers with dynamic number of compositions

In the experiments, we have used both static AAS and our new DynAAS algorithm. Static AAS restarts the search from scratch every time a change is detected. Figure 4 shows the performance of DynAAS and static AAS in terms of mean values of normalized *instantaneous errors* vesus rate of change. The graph shows that the error rate of DynAAS is significant lower than AAS for low changed rates and increased for larger values of σ. This can be explained as the the change rate is greater than the adaptive rate at which DynAAS can handle. However it shows that DynAAS offers significant reduction in solution errors if the change rate is reasonable. This reduction is greater than 50% for $\sigma \leq 8$. In other words, for the current setup, as long as there are no more than 8 constraints added or removed at each processing cycle then DynAAS gives twice of the level of solution accuracy over static AAS.

VI. Conclusions

This paper describes a new extension of AAS called DynAAS for dynamic environment where unexpected events and changes can happen. DynAAS allows different centralized CSP solvers to be incorporated into the algorithm hence make it more flexible in solving local CSP problems. Experiments show the applicability of AAS and DynAAS in solving the QoS guarantees with multiple compositions of Web services, and the improvement of DynAAS over AAS in terms of efficency for reasonal changes. Our future work focuses on a framework which allows agents to balance between privacy and information revealed during communications, as a part of J4WSM toolkit. We believe this direction is promising to improve the practicability of our approach in the real Web service environment.

References

[1] X. Gu, K. Nahrstedt, R. Chang, and C. Ward. Qos-assured service composition in managed service overlay networks, 2003.

[2] P. Hubbe and E. Freuder. An efficient cross-product representation of the constraint satisfaction problem search space. In *AAAI-92*, pages 421–427, 1992.

[3] M. C. Jaeger, G. Rojec-Goldmann, and Mühl. QoS aggregation for service composition using workflow patterns. In *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 149–159, Monterey, California, USA, 2004. IEEE CS Press.

[4] Y. Liu, A. H. Ngu, and L. Z. Zeng. Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004. ACM Press.

[5] *ILog JSolver home page*. www.ilog.com/products/jsolver/index.cfm.

[6] *NSolver home page*. www.cs.cityu.edu.hk/ hwchun/nsolver/.

[7] M. C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. In *Artificial Intelligence Journal Vol.161*, pages 25–53, New York, NY, USA, 2005. ACM Press.

[8] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Intelligent domain splitting for CSPs with ordered domains. In *Principles and Practice of Constraint Programming*, pages 488–489, 1999.

[9] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *AAAI/IAAI*, pages 917–922, 2000.

[10] G. Verfaillie and T. Schiex. Dynamic backtracking for dynamic constraint satisfaction problems. In *Proceedings of the ECAI'94 Workshop on Constraint Satisfaction Issues Raised by Practical Applications, Amsterdam, The Netherlands*, pages 1–8, 1994.

[11] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *National Conference on Artificial Intelligence*, pages 307–312, 1994.

[12] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.

Proceedings of the 2006 IEEE/WIC/ACM International Conference on
Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)
0-7695-2749-3/06 $20.00 © 2006 IEEE