A Novel Chemistry-inspired Approach to Efficient Coordination of Multi-mission Networked Objects

Mahmoud ElGammal

The Bradley Department of
Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
Email: gammal@vt.edu

Mohamed Eltoweissy*

Department of Computer and Information Sciences
Virginia Military Institute
Lexington, Virginia 24450
Email: eltoweissymy@vmi.edu

***Abstract*—In this paper we present a chemistry-inspired approach for coordinating networked objects in pervasive computing environments built on the concept of *chemical affinity*. Our thesis is that by paralleling the model of interaction that takes place among atoms during a chemical reaction, a form of collective intelligence emerges among the objects in the network enabling them to achieve a common global objective while relying solely on preferences expressed on an individual basis. The main contribution of this paper is a novel implementation of a highly-parallelized chemical reaction execution engine that uses message passing to optimize reactant selection for multiple reaction rules simultaneously. In our method, objects in the chemical domain are represented using a probabilistic factor graph, where inter-reactant affinities are encoded in the factor nodes to guide bond formation among reactants. The problem of associating reactants with reaction rules is modeled as a Maximum-a-Posteriori (MAP) assignment problem, which we solve using the Max-Product Belief Propagation algorithm, allowing us to efficiently obtain a reactant-to-reaction assignment that maximizes the number of concurrent reactions. To evaluate our approach, we use simulation to assess the performance of the reaction execution engine in terms of execution speed and solution quality. Finally, we use the problem of resource-constrained task assignment among heterogeneous robots as a case study to present a concrete application of our approach.**

***Keywords*–Nature-inspired computing; Internet-of-Things; Pervasive computing; Belief propagation; Computational Chemistry.**

I. INTRODUCTION

With the current surge in smart computing applications, researches are increasingly turning to nature-inspired computing models for ideas on how to deal with the highly dynamic nature of this new computing paradigm. In [1] we proposed a new approach to network configuration and management in pervasive computing systems inspired by the *chemical affinity* concept, which we coined *C_2A_2: Chemistry-inspired, Context-Aware, and Autonomic Management System for Networked Objects*. The concept behind C_2A_2 is that physical and logical components of the network are mapped to the chemical domain using a layered abstraction model. Once represented using the chemical metaphor, reaction rules are then defined to specify how reactants in the chemical domain are allowed to interact with each other, which would eventually lead to implications on the actual network. C_2A_2 relied on a *reaction execution engine* that was responsible for deciding which reaction rules may be fired and which reactants should be consumed by them. The work presented herein serves as a more in-depth discussion of a significantly improved implementation of the reaction execution engine previously introduced in [1].

In our new approach, reactants and reaction rules in the chemical domain are modeled using a probabilistic factor graph, where factor nodes encode affinities between each pair of reactants, affinities between reactants and reaction rules, as well as the different constraints needed to ensure that the resulting solution constitutes a valid reactant-to-reaction assignment. The graph is constructed such that the optimal assignment of reactants to reaction rules can be obtained by solving the Maximum-a-Posteriori assignment problem [2] on the graph, which we solve by passing carefully designed messages over the graph according to the Max-Product Belief Propagation algorithm [3]. A key advantage of this approach is its ability to find a solution that maximizes the number of reaction rules that can be satisfied simultaneously, which makes it particularly suited for multi-mission pervasive computing applications.

The remainder of the document is organized as follows. In Section II, we survey some notable related works from the literature and provide an overview of the probabilistic graphical modeling techniques we rely on in later sections. In Section III, we present the implementation of our reaction execution engine. In Section IV, we analyze the performance of our approach using simulation, and validate its efficacy by applying it to a more concrete case study. Finally, in Section V, we present our conclusion and discuss future work.

II. BACKGROUND AND RELATED WORK

In C_2A_2 we leverage ideas from the fields of *bio-inspired computing* and *machine learning*. In particular, we build our work upon some of the established concepts and algorithms belonging to the subfields of *chemistry-inspired*

*Also Affiliate Professor, The Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA.

computing and *probabilistic graphical modeling*[1] in those two research disciplines, respectively. In this section we present a rudimentary overview of these topics and discuss some important related work.

A. Chemistry-inspired Computing

A common feature among most, if not all, natural computing models that draw their inspiration from the chemical reaction metaphor, is that the system state is represented as a fluid in which reactants of different types move freely and interact with each other according to predefined reaction rules. Developing concrete applications based on this concept requires mature models of computation that can be used to encode real-life problems using the chemical formalism and describe programs to solve them, as well as runtime systems that can actually execute these programs. The former field of study has seen significant research activity, ushered by the Γ language by Banâtre et al. [4] and continued through various other works such as the Chemical Abstract Machine by Berry et al. [5], the Molecular Dynamics model by Bergstra et al. [6], Membrane Computing (P Systems) by Păun [7], and more recently in the Biochemical Tuple Spaces model by Viroli et al. [8]. Some of these works contributed incremental improvements over previous models while others offered entirely new approaches, but all have served to present the chemical metaphor as a mature and viable option for modeling computational processes, especially for applications where concurrency and self-organization are two desirable characteristics. However, despite the progress on this front, not as much attention has been given to the runtime systems on which chemical computing models can be executed [9].

The earliest runtime system for a chemical machine is perhaps the one described in [4], where an implementation of the Γ language on a massively parallel machine (aka the Γ-machine) is proposed. In order to evaluate a Γ program, the runtime system has to perform two tasks: (a) search for reactants that satisfy reaction conditions (in other words, determine which reactions to fire), and (b) applying the actions associated with fired reactions on the system. It can easily be shown that a roughly similar breakdown of tasks would also apply to any other runtime, not just the Γ-machine. The first task requires solving an $\mathcal{NP}$-hard optimization problem, and with C_2A_2, we attempt to put forward a practical, efficient, and scalable solution to this problem.

Existing runtime systems employ different approaches to address this problem. One approach can be described as the *search-and-match* approach, and it usually relies on some data structure that stores information about reactants and reaction rules where a search algorithm is then used to determine which reactions to fire based on the satisfiability of their premise. The implementation proposed for the Γ-machine in [4] falls under this category. However, it can be considered more of a proof of concept as it assumes a number of processors equal to the number of reactants, which would be faced by strict scalability limits in reality. Additionally, it only considers one form of reactions (more specifically, reactions that take exactly two input values and produce two output values), which would impose further constraints on the practicality of this approach. More efficient methods belonging to this category have also been proposed, such as The Chemical Machine by Rajcsányi et al. [10], which relies on the more sophisticated RETE pattern-matching algorithm [11].

Another approach that is mainly used for simulation but is also used in some runtime systems is the *computational chemistry* approach. Methods belonging to this category rely on algorithms that have long been used by theoretical chemists to solve many quantitative chemical problems using simulation with acceptable accuracy. Several algorithms have been developed under this category and which have been improving in efficiency over time, such as Gillespie's First Reaction Method [12], the Next Reaction Method by Gibson et al. [13], Slepoy et al.'s constant-time Monte Carlo algorithm for simulating biochemical reaction networks [14], ALCHEMIST [15], and others. These methods offer a statistically correct depiction of the evolution of species concentration in a chemical solution over time, which is necessary in applications that rely on accurate simulation of the laws of chemical kinetics.

These two approaches have different points of strength and weakness. The search-and-match approach can be used to model the behavior of a self-organizing system in terms of microscopic interactions among its lowest-level components, which makes it a more versatile tool for modeling a wide range of applications. On the downside, it offers limited control on the macroscopic behavior of the whole system [16]. The computational chemistry approach on the other hand offers greater control over the macroscopic behavior of the system, which allows for better overall stability and predictability, but only if the target application lends itself easily to this approach, such as the case studies given in [15], [16], [17], [18].

In C_2A_2 we utilize the concept of *chemical affinity* to express the mutual attraction force between a reactant and another reactant or reaction rule, and we present an approach for reconciling the inevitably conflicting preferences of reactants to form bonds among themselves. By doing so, we aim to combine some of the advantages of the two approaches mentioned above, where the macroscopic behavior of the system is derived from the individual inclinations of reactants at the microscopic level. Furthermore, our approach maximizes the number of reaction rules that can be satisfied simultaneously, making it particularly efficient for applications requiring the execution of concurrent processes.

B. Probabilistic Factor Graphs

Probabilistic graphical models [3] provide a framework that enables us to encode our knowledge of how a real-world system works in a compact, machine-readable format, amenable to the application of various reasoning algorithms. Being declarative by nature, such modeling method has the advantage of separating the domain-specific knowledge needed to construct the graph from the reasoning algorithms that can be applied to it. This allows the development of

[1] For a thorough review of probabilistic graphical models, the reader is referred to [3].

a whole host of reasoning algorithms that can manipulate the model and answer various queries about it despite being agnostic of the semantics of the real-world system it represents.

A probabilistic graphical model consists of a number of nodes, each representing a random variable, which in turn represents some aspect of the modeled system. The nodes are connected using either directed or undirected edges (as in Bayesian and Markov networks, respectively), which express how a pair of random variables interact. Assuming that we have prior knowledge of how the random variables in the graph are probabilistically interdependent (or not, in case of independent variables), our job is to reason about the most likely values of one or more variables, possibly after having observed the values of some others. In order to do that, we are often interested in calculating the joint distribution over the possible values of all random variables in the graph or a subset thereof, which can quickly become an intractable problem. For instance, in a graph containing *N* binary random variables, the space of possible assignments to these variables is of size 2^N.

However, since most complex distributions usually contain independencies among many of their dimensions, and since the graph is essentially a representation of these independencies, we can exploit the structure inherent in the distribution by breaking it up into smaller *factors*, each having just a subset of interdependent variables in its scope. A factor ϕ_i with scope D_i is a function that maps each possible assignment of the variables in D_i to a nonnegative value, essentially signifying the affinity of the random variables toward each possible joint assignment. The graph structure then becomes a factorization of the overall joint distribution P over all variables in the domain $\mathcal{X} = \{X_1, \ldots, X_n\}$, which can now be expressed more compactly as the product of these factors:

$$P(\mathcal{X}) = \frac{1}{Z} \prod_i \phi_i(D_i) \quad (1)$$

where Z is a normalization constant (also known as the *partition function*), added in order to make $P(\mathcal{X})$ a valid probability distribution function.

We are specifically interested in *factor graphs* – a special type of undirected probabilistic graphical models that make the factorization explicit in the graph structure. A factor graph is a bipartite graph consisting of one group of nodes that represent factors, another that represents random variables, and edges that exist only between a factor node and nodes of variables that lie in its scope. In the following sections, we discuss how we optimize reaction selection in C_2A_2 by constructing a factor graph for the system and transforming the problem into one of the well-known inference problems applicable to probabilistic graphical models, which we cover next.

C. The MAP Assignment Problem

In the MAP (*Maximum a Posteriori*) assignment problem we try to answer the following question: given a set of random variables $\mathcal{X} = \{X_1, \ldots, X_n\}$, of which a subset E (the evidence) is observed to have the value e, what is the *joint* assignment for the remaining variables $W = \mathcal{X} - E$ such that $P(W|E = e)$ is maximized?

An important thing to note is that the correct answer to this query is quite different from what could be obtained by selecting the most likely assignment for the individual members of W given the evidence, which could possibly yield an invalid solution. For instance, while the most likely independent assignments for a pair of variables $X, Y \subset W$ given evidence $E = e$ may be x and y, respectively, it is possible that these two assignments can never co-occur given the same evidence. Expressed more formally, there is no guarantee that $\{\mathrm{argmax}_X P(X|e), \mathrm{argmax}_Y P(Y|e)\}$ and $\mathrm{argmax}_{X,Y} P(X, Y|e)$ are equivalent. This adds a significant number of constraints that need to be satisfied by the solution, making MAP assignment an $\mathcal{NP}$-hard problem [3].

MAP inference continues to be a useful tool for solving problems in various domains, such as computer vision, speech recognition, noisy-channel coding, medical diagnosis, and many others. In all of these problems, we rely on partially observed data in conjunction with some predefined assumptions about the data model in order to estimate an unobserved quantity. As will be explained in more detail shortly, in C_2A_2 we encode the *affinities* between the different objects in the network as well as the reaction rules that dictate how objects interact with each other into the factor graph devised for the system. The graph then becomes the input to a MAP assignment problem whose output determines which reactions are fired and which reactants are consumed thereby.

Owing to its intractability in large graphs, MAP inference is usually solved using one of several available heuristic algorithms. The one we are concerned with here is *Belief Propagation* [19], which we briefly discuss next before moving on to our system implementation.

D. Belief Propagation

Belief Propagation [19] (BP) is a message-passing algorithm for performing probabilistic inference on graphical models. Given a set of factors Φ defined over a set of random variables $\mathcal{X}$, we start by constructing a *cluster graph*, which is an undirected graph where each node (or cluster) *i* is associated with a subset of variables $C_i \subseteq \mathcal{X}$, and each pair of clusters C_i and C_j with shared variables are connected by an edge associated with a *sepset* $S_{i,j} \subseteq C_i \cap C_j$. The algorithm then progresses through a number of iterations, during which each cluster node transmits to each of its direct neighbor clusters its beliefs about the variables in their shared sepset. A message from C_i to C_j incorporates evidence collected by C_i about the variables in the sepset $S_{i,j}$ from all its direct neighbors except C_j (which serves to avoid reinforcing a prior incorrect belief by C_j). Provided that the graph satisfies certain properties that will be discussed later, beliefs at neighboring clusters concerning their shared sepset converge after a finite number of iterations, at which point the graph is said to have become *calibrated*. The final step consists of decoding the calibrated beliefs at every cluster to obtain the solution to our inference problem.

BP has different variants for solving different inference problems. We focus our attention on Max-Product Belief Propagation (MPBP), which can be used to solve the MAP assignment problem. In MPBP, the belief computed at cluster C_i is the *max-marginal* for the variables in its scope, which is a function that maps each possible configuration of the variables in $Scope[C_i]$ to the unnormalized probability of the most likely joint assignment consistent with it. If the cluster graph is actually a tree, then the MAP problem can be solved exactly and the max-marginals can be computed in polynomial time. Moreover, if there is only one unique optimal assignment to the unobserved nodes, then decoding the max-marginals to obtain the solution is trivial and can be performed in $O(N)$ [3]. Under different circumstances, the solution is only approximate, and certain measures have to be taken to guarantee convergence and simplify the final decoding step. We discuss the application of BP to our problem in more detail as well as the exact format of the messages passed on the graph in the next section where we present our system implementation.

III. System Implementation

As in other chemical computing models (e.g., [5], [20], [7]), the state of the system in C_2A_2 is analogous to a chemical solution in which reactants move freely and interact with each other according to predefined reaction rules. In an actual chemical solution, Brownian motion is responsible for causing reactants to come in contact with each other, where they form different bond types at various rates that depend on their relative concentrations, the presence of certain catalysts, among other factors. In a chemical computing system, this is emulated via an algorithm that decides which reactions are triggered at what times and which reactants are consumed by them (e.g., [12], [14], [15]). In this section, we put forward an implementation of one such algorithm.

A. Overview

The reaction execution engine in C_2A_2 is an implementation of a P System [7]. Reactions in the forward direction take the form $ca \rightarrow cu$, while reverse directions take the form $cu \rightarrow ca$. In both forms, a and u are reactant multisets, and c is a catalyst that may optionally be required in order for the reaction to be triggered.

The problem of assigning reactants to forward reactions can be seen as a clustering problem in which reaction rules play the role of exemplars while reactants (atoms and molecules) play the role of cluster members. However, one property of P Systems necessitates a slight departure from traditional clustering, which is that reactions must be performed in a maximally parallel way. Consequently, reactants are assigned to rules until no further assignments are possible, which imposes the consequence that a single rule may assume exemplarity of multiple cluster instances, all of identical structure, where the members of each instance have a one-to-one correspondence to the elements in the left-hand side multiset in the forward reaction form, a. This reformulated clustering problem can be solved by constructing a factor graph as the one shown in Figure 1, where the final reactant/reaction assignments are decided through the iterative exchange of *affinity* values according to the BP algorithm.

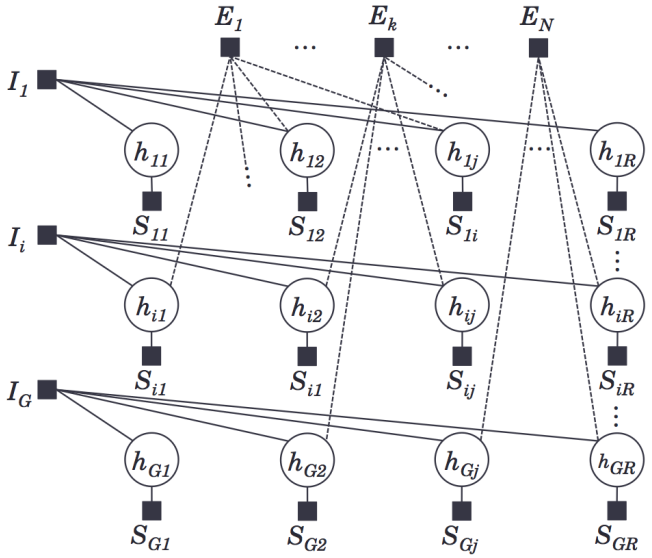

Figure 1. Factor graph of reformulated clustering problem for the reaction execution engine in C_2A_2.

B. Constructing the Factor Graph

In constructing the factor graph for our problem, we have extended the binary variable model for the Affinity Propagation (AP) clustering algorithm [21] presented in [22]. We assume that the system contains N reactants and R reaction rules, and the goal is to match different subsets of these N reactants with reaction rules such that each subset (henceforth called *reactant group* or just *group* for short) is congruent with the left-hand side multiset of the reaction it is matched with, and no single reactant is involved in more than one reaction.

As illustrated in Figure 1, the factor graph is 2-dimensional and is constructed such that reaction rules are organized horizontally and are indexed by the variable $j \in \{1 \dots R\}$, while symbols representing reactant groups are organized vertically and are indexed by the variable $i \in \{1 \dots G\}$. Reactant groups represent the elements of the power set over all reactants, which has a size of 2^N. Obviously, this can be prohibitively large. In practice, however, reactant groups that do not appear in any reaction rules as well as those that do not satisfy a minimum affinity threshold among its members can be eliminated, reducing the size of the set by several orders of magnitude. The h_{ij} nodes are binary variables, which if set to 1 at the end of executing the message-passing algorithm indicate that group i is to be consumed by reaction j.

Because a single reactant may belong to multiple groups, if group i is assigned to rule j, we must guarantee that: (1) group i is not simultaneously assigned to any other rule $k \neq j$, and (2) none of the reactants in group i belong to another group consumed by a reaction. These two constraints are enforced by adding the I_i and $E_{k \in \{1 \dots N\}}$ factor nodes, respectively. The S_{ij} factor nodes provide the initial affinity

values between each group i and reaction j. By applying equation (1) to the factor graph in Figure 1 (and ignoring the Z constant), the objective function maximized by MPBP becomes:

$$\underset{mpbp}{f(\{h_{ij}\})} = \prod_{i=1}^{G}\prod_{j=1}^{R} S_{ij}(h_{ij}) \cdot \prod_{i=1}^{G} I_i(h_{i:}) \cdot \prod_{k=1}^{N} E_k(h^k_{::})$$

where $h^k_{::}$ denotes all the h_{ij} variables in the scope of the E_k factor. Because there are often several advantages to optimizing $log(f_{mpbp})$ instead [3], we use a different version of MPBP that maximizes the summation of factors in log-space instead of their product. The algorithm is called Max-Sum Belief Propagation (MSBP) which is the one we use in our implementation. The final objective function hence becomes:

$$\underset{msbp}{f(\{h_{ij}\})} = \sum_{i=1}^{G}\sum_{j=1}^{R} S_{ij}(h_{ij}) + \sum_{i=1}^{G} I_i(h_{i:}) + \sum_{k=1}^{N} E_k(h^k_{::}) \quad (2)$$

The three factor types work together to produce a solution that maximizes the sum of affinities over all possible $(group, reaction)$ pairs, but without violating any of the stipulated constraints. Invalid solutions are eliminated by assigning a value of $-\infty$ to a factor when the constraint it is associated with is violated, guaranteeing a sub-optimal result for the objective function.

C. Message Updates

We now derive the messages passed between the different nodes in the factor graph. In the max-sum algorithm, a message passed from one node to another can have one of the two forms shown in equations (3) and (4) depending on whether it flowed from a variable node v to a factor node f or in the opposite direction, respectively. In equation (3), $\mathcal{N}(v)$ is the set of all factor nodes connected to v, while in equation (4) $\mathcal{X}_f$ is the scope of factor f.

$$\mu_{v\to f}(val_v) = \sum_{g\in\mathcal{N}(v)\setminus f} \mu_{g\to v}(val_v) \quad (3)$$

$$\mu_{f\to v}(val_v) = \max_{\mathcal{X}_f\setminus v}\{f(\mathcal{X}_f) + \sum_{w\in\mathcal{X}_f\setminus v} \mu_{w\to f}(val_w)\} \quad (4)$$

For each variable node, a message has to be exchanged to and from all neighbor nodes for each possible assignment of the variable. Since all variables in the factor graph in Figure 1 are binary, [22] shows that we could compute the difference between the two messages associated with the two possible assignments (e.g., $\mu_{v\to f}(v^1) - \mu_{v\to f}(v^0)$) such that only one message need be sent per neighbor in each direction. Figure 2 shows a fragment of the full factor graph that illustrates all the messages involved in computing the final value of a single h_{ij} variable. We now discuss the role played by each of these messages in our application as well as the full derivation for the max-sum BP algorithm.

1) Message from S_{ij} factor: The S_{ij} factor represents the *reward* gained by the system when reactant group i is assigned to reaction j. The s_{ij} message, which is sent from the S_{ij} factor node to its associated h_{ij} variable node, has a constant value that represents the affinity between reactant group i and reaction j. If group i matches the input multiset

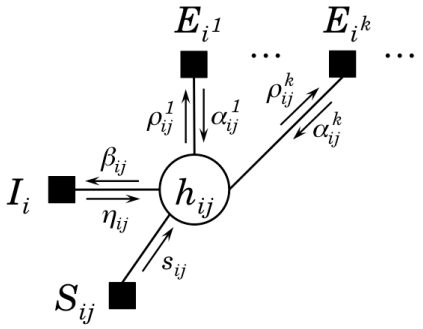

Figure 2. Factor graph fragment for computing the affiliation of reactant group i with reaction rule j.

of rule j and the group is actually assigned to the reaction, then $S_{ij}(h_{ij})$ is set to the average affinity between the reaction and the reactants in the group. Otherwise, the pair is not candidate for matching and the factor yields a value of $-\infty$. In equation (6), the average is used to normalize the resulting affinity value, since reactant groups can be of different sizes.

$$S_{ij}(h_{ij}) = \begin{cases} s_{ij}h_{ij} & \text{if group}_i \equiv \text{rule}_j \\ -\infty & \text{otherwise} \end{cases} \quad (5)$$

$$s_{ij} = avg_k\{\mathit{aff}(i^k, j)\} \quad (6)$$

2) Messages from/to I_i factor: The I_i factor guarantees that reactant group i is assigned to at most one reaction. Note that in equation (7), we use 0 instead of 1 when the constraint is satisfied since we are operating in log-space.

$$I_i(h_{i:}) = \begin{cases} 0 & \text{if } \sum_{j=1}^{R} h_{ij} \le 1 \\ -\infty & \text{otherwise} \end{cases} \quad (7)$$

This is achieved by exchanging the η_{ij} and β_{ij} messages between the factor node and the h_{ij} node for each reaction. The derivation provided below for these two messages is similar to the original derivation for the binary model for AP in [21], but it accounts for the different structure of the factor graph in our problem, where each h_{ij} node could be connected to a varying number of E_k factors. Due to this, we use the average of the incoming α_{ij} messages in the equations below instead of the summation.

- β_{ij} message (using equation (3)):

$$\begin{aligned}\beta_{ij}(h_{ij}=1) &= \mu_{h_{ij}\to I_i}(1) \\ &= \mu_{S_{ij}\to h_{ij}}(1) + avg_k\{\mu_{E^i_k\to h_{ij}}(1)\} \\ &= s_{ij}(1) + avg_k\{\alpha^k_{ij}(1)\}\end{aligned}$$

  Similarly:

$$\beta_{ij}(h_{ij}=0) = s_{ij}(0) + avg_k\{\alpha^k_{ij}(0)\}$$

  Given that each message is computed as the difference between its formulae for the two possible assignments of h_{ij}, we get:

$$\beta_{ij} = \beta_{ij}(1) - \beta_{ij}(0) = s_{ij} + avg_k\{\alpha^k_{ij}\} \quad (8)$$

- η_{ij} message (using equation (4)):

$$\begin{aligned}\eta_{ij}(h_{ij}=1) &= \mu_{I_i\to h_{ij}}(1)\\ &= \max_{j'\neq j}\{I_i(h_{i1},\ldots,h_{ij}=1,\ldots,h_{iR}) + \sum_{h_{ij'}\in\mathcal{N}(I_i)\setminus h_{ij}} \mu_{h_{ij'}\to I_i}(h_{ij'})\}\\ &= \max_{j'\neq j}\{I_i(h_{i1},..,h_{ij}=1,..,h_{iR}) + \sum_{j'\neq j}\beta_{h_{ij'}}(h_{ij'})\}\end{aligned}$$

Since we are assuming that $h_{ij} = 1$, the only configuration that would maximize $\eta_{ij}(1)$ is if every other $h_{ij',j'\neq j}$ was set to 0, otherwise the group would be assigned to more than one reaction and I_i would yield $-\infty$ (equation (7)). $\eta_{ij}(1)$ then becomes:

$$\eta_{ij}(h_{ij}=1) = \sum_{j'\neq j}\beta_{h_{ij'}}(0)$$

For $h_{ij}=0$, the configuration is only valid when a unique $h_{ij',j'\neq j}=1$, in which case the $I_i(h_{i:})$ term becomes 0, and the formula for $\eta_{ij}(h_{ij}=0)$ becomes a maximization over all such configurations:

$$\eta_{ij}(h_{ij}=0) = \max_{j'\neq j}\{\beta_{ij'}(1) + \sum_{l\notin\{j,j'\}}\beta_{il}(0)\}$$

Finally, η_{ij} is obtained by subtracting $\eta_{ij}(0)$ from $\eta_{ij}(1)$:

$$\eta_{ij} = -\max_{j'\neq j}\{\beta_{ij'}\} \tag{9}$$

3) Messages from/to E_k factor: The E_k factor implicitly guarantees that if a reactant group is assigned to a reaction, then no other group with any shared reactants is assigned to a reaction. This is achieved by ensuring that at most one of the h_{ij} variables attached to each E_k factor is set to 1:

$$E_k(h_{::}) = \begin{cases}0 & \text{if } \sum_{(i,j)^k} h_{ij}^k \leq 1\\ -\infty & \text{otherwise}\end{cases} \tag{10}$$

where $(i,j)^k$ are the $(group, reaction)$ indices of all the h_{ij} variables attached to E_k ($k\in\{1\ldots N\}$). The E_k factor exchanges two messages with each variable it is connected to: the α_{ij}^k message represents the accumulated evidence of how appropriate it would be for the k^{th} reactant in group i to partake in reaction j, while the ρ_{ij}^k message represents the accumulated evidence for the willingness of reaction j to consume the reactant, which has to take into consideration the availability of all other reactants involved in the group. In the following equations, the notation E_{i^k} denotes the E factor associated with the k^{th} reactant in group i.

- ρ_{ij}^k message (using equation (3)):

$$\begin{aligned}\rho_{ij}^k(h_{ij}=1) &= \mu_{h_{ij}\to E_{i^k}}(1)\\ &= \mu_{S_{ij}\to h_{ij}}(1) + \mu_{I_i\to h_{ij}}(1) + \underset{k'\neq k}{avg}\{\mu_{E_{i^{k'}}\to h_{ij}}(1)\}\\ &= s_{ij}(1) + \eta_{ij}(1) + \underset{k'\neq k}{avg}\{\alpha_{ij}^{k'}(1)\}\end{aligned}$$

and it can easily be shown that $\rho_{ij}^k(h_{ij}=0)$ would have a similar formula to $\rho_{ij}^k(h_{ij}=1)$, yielding the following for ρ_{ij}^k:

$$\rho_{ij}^k = s_{ij} + \eta_{ij} + \underset{k'\neq k}{avg}\{\alpha_{ij}^{k'}\} \tag{11}$$

- α_{ij}^k message (using equation (4)):

Let H be the set of h variables connected to E_{i^k} except h_{ij}: $H = \mathcal{N}(E_{i^k})\setminus h_{ij}$. Since $h_{ij}=1$, the only configuration that ensures that the reactant is not assigned to multiple reactions is when $h=0\,\forall\, h\in H$:

$$\begin{aligned}\alpha_{ij}^k(h_{ij}=1) &= \mu_{E_{i^k}\to h_{ij}}(1)\\ &= \max_{h\in H}\{E_{i^k}(h_{ij}=1,H) + \sum_{h\in H}\mu_{h\to E_{i^k}}(h)\}\\ &= \max_{h\in H}\{E_{i^k}(h_{ij}=1,H=\{0\}) + \sum_{h\in H}\rho_{h\to E_{i^k}}(0)\}\\ &= \max_{h\in H}\{0 + \sum_{h\in H}\rho_{h\to E_{i^k}}(0)\} = \sum_{h\in H}\rho_{h\to E_{i^k}}(0)\end{aligned}$$

For $h_{ij}=0$, the configuration is valid when all variables in H are set to 0 as in the previous case, or when exactly one variable in H is set to 1:

$$\begin{aligned}\alpha_{ij}^k(h_{ij}=0) &= \mu_{E_{i^k}\to h_{ij}}(0)\\ &= \max_{h\in H}\{E_{i^k}(h_{ij}=0,H) + \sum_{h\in H}\mu_{h\to E_{i^k}}(h),\}\\ &= \max_{h\in H}\{E_{i^k}(h_{ij}=0,h_1=0,\ldots,h_{|H|}=0) + \sum_{h\in H}\rho_{h\to E_{i^k}}(0),\\ &\quad E_{i^k}(h_{ij}=0,h_1=1,h_2=0,\ldots,h_{|H|}=0) + \rho_{h_1\to E_{i^k}}(1) + \sum_{h\in H\setminus h_1}\rho_{h\to E_{i^k}}(0),\ldots,\\ &\quad E_{i^k}(h_{ij}=0,h_1=0,h_2=0,\ldots,h_{|H|}=1) + \rho_{h_{|H|}\to E_{i^k}}(1) + \sum_{h\in H\setminus h_{|H|}}\rho_{h\to E_{i^k}}(0)\}\end{aligned}$$

In the equation above, $E_{i^k}(h_{::})$ is 0 for any valid configuration. By using the fact that $a - max(b,c) = min(a-b, a-c)$, the formula for α_{ij}^k obtained by subtracting $\alpha_{ij}(0)$ from $\alpha_{ij}(1)$ becomes:

$$\alpha_{ij}^k = min\{0, \min_{(g,r)\in(i,j)^k\setminus(i,j)}\{-\rho_{gr}^k\}\} \tag{12}$$

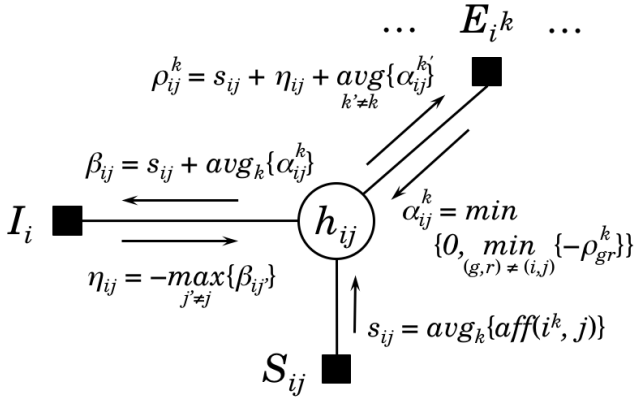

Figure 3. Messages used to compute the affiliation of reactant group i with reaction j using max-sum belief propagation.

This concludes the derivation of all messages passed over the factor graph, which are summarized in Figure 3. An important remark regarding the formulae for the η_{ij} and α_{ij}^k messages (equations (9) and (12)) produced by the I_i and E_{i^k} factors, respectively, is that they have a

complexity of $O(N)$ in the number of possible values for the input messages that must be processed by the factor. Typically, this step would be of exponential complexity in the number of possible assignments of the variables in the scope of the factor, but due to the constraints employed in our problem, the global solution search space could be pruned significantly via early elimination of invalid solutions locally at each factor. This should work to our advantage in terms of the overall execution cost for this implementation.

D. Constructing the Cluster Graph

As mentioned in Section II-D, message passing in belief propagation takes place over a *cluster graph* derived from the factor graph representation rather than directly on the latter. Different cluster graphs can be generated for the same factor graph, but for a cluster graph to be valid, it must satisfy two properties [3]:

- The *family-preservation* property, which stipulates that for each factor ϕ in the original factor graph there must exist a cluster C in the cluster graph such that $Scope[\phi] \subseteq Scope[C]$, and
- The *running-intersection* property, which requires that for any two clusters containing a variable v, there is exactly one path between the two clusters over which information about v can be propagated.

A certain class of cluster graphs knows as *Bethe cluster graphs* is guaranteed to satisfy these two properties [3]. A Bethe cluster graph is a bipartite graph where a univariate cluster is added for each variable in the factor graph, and factors are represented by multivariate clusters whose scopes include all variables in their respective factors. Edges are added only between a univariate cluster and the multivariate clusters whose scopes include the variable in question. Generating such cluster graph from the factor graph in Figure 1 is straightforward. The result is shown in Figure 4, which can be used to verify that both the family-preservation and running-intersection properties are satisfied.

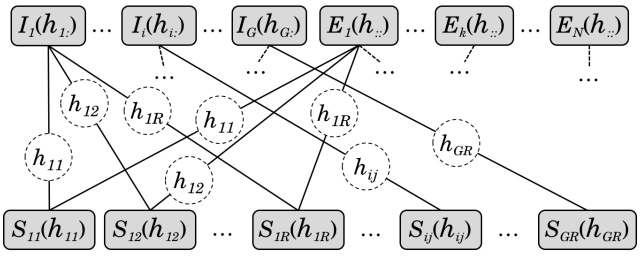

Figure 4. Cluster graph used for passing messages according to max-sum belief propagation. Univariate clusters representing each S_{ij} factor appear in the bottom row, while multivariate clusters for the I_i and E_k factors appear in the top row. Edge labels indicate the variable about which information is passed along the edge.

1) Dealing with non-convergence: As can be seen in Figure 4, the graph is not loop-free (e.g., the loop $S_{11} - I_1 - S_{12} - E_1 - S_{11}$), which means that belief propagation is not guaranteed to converge, or to obtain the optimal variable assignment when it does converge. Obviously, the problem of non-optimality cannot be completely eliminated. However, by ensuring that the cluster graph satisfies the running-intersection property, the quality of the approximate solution is improved as we verify in detail in Section IV. The problem of non-convergence, on the other hand, is dealt with by employing the following techniques:

- *message damping*, where the value of a message in any given iteration of the algorithm is computed as the weighted average of the new message value and its value from the previous iteration:
$$\mu_{t+1} \leftarrow d \cdot \mu_t + (1-d) \cdot \mu_{t-1}, \quad 0 < d < 1$$
This helps reduce the chances of the message passing algorithm to oscillate indefinitely between two possible configurations.
- Terminating the message passing algorithm after having observed no improvement in the objective function (equation (2)) for a finite number of iterations. Doing so guarantees halting, but it also means that in some cases the algorithm may never find a solution. However, experimental results have shown that this only happened with an acceptably low probability, and can be remedied satisfactorily as we show in Section IV.

2) Optimizing performance: A significant reduction in the number of messages passed over the graph can be achieved by considering the fact that, typically, each reaction rule is compatible only with a small subset of the available reaction groups. This means that a large percentage of the h_{ij} variables in Figure 1 could never be set to 1 in a valid solution. Consequently, the beliefs about such variables in the cluster graph should remain constant throughout the execution of the algorithm, with the likelihood of $h_{i,j} = 0$ being 1 and that of $h_{ij} = 1$ being 0 (or 0 and $-\infty$ in log-domain, respectively). An optimization that we applied to our implementation in order to eliminate unnecessary message exchange with such variables is to initialize the beliefs about them in the multivariate clusters that include them in their scopes to $belief(h_{ij}) = \{0 \rightarrow 0, 1 \rightarrow -\infty\}$ while removing the univariate clusters that represent them. This resulted in significantly faster execution times without affecting the quality of the obtained solution.

Another potentially time-consuming operation is the generation of candidate reaction groups that may be assigned to a reaction. In the case where there is a large number of instances of one or more reactants, a combinatorial explosion in the number of candidate reactant groups may occur, which can add a considerable time penalty. In order to circumvent this problem, we have used the following strategies:

- Amortizing the cost of generating candidate reactant groups by caching previous candidates and updating them upon creating or destroying reactant instances, and
- Considering only the top n reactants with the highest affinity values toward a reaction rule when updating the reactant groups candidate for consumption by the rule.

E. Updating the Cluster Graph

Because of the dynamic nature of applications modeled using the chemical metaphor, it is necessary to modify the

structure of the cluster graph when certain events take place, such as:

- a reactant is added or removed from the system.
- a reaction is rule is activated or deactivated due to its inputs being satisfied or unsatisfied, respectively.
- a new reaction rule is defined in the system or an existing rule removed.
- inter-reactant or reactant-reaction affinities change.

Even when no external changes are exerted on the system (such as the introduction of new reactants or reaction rules) most of these events take place by merely executing a reaction rule. Instead of reconstructing the cluster graph from scratch when one of these events occur, the graph structure is only amended accordingly. The system is said to have reached *equilibrium* when no more reactions can be executed. The pseudocode for our reaction execution engine is shown below.

```
1:  procedure main()
2:    rules ← findRulesWithSatisfiedInputs()
3:    reactantGroups ← φ
4:    for each rule in rules do
5:      reactantGroups ← reactantGroups ∪
            computeReactantGroupsForRule(rule)
6:    end for
7:    graph ← constructBetheClusterGraph
            (rules, reactantGroups)
8:    while true do
9:      rules ← selectReactionRulesToExecute(graph)
10:     if rules = φ then
11:       // System reached equilibrium
12:       return
13:     end if
14:     for each rule in rules do
15:       rule.execute()
16:     end for
17:     updateAffinitites()
18:     graph ← amendGraph(graph)
19:   end while
20: end procedure
21: procedure selectReactionRulesToExecute(graph)
22:   // Initialize messages
23:   for each cluster in graph.multivarClusters() do
24:     for each var_hij in cluster.scope() do
25:       if isCompatible(group_i, rule_j) then
26:         μ_{S_ij→cluster} ← 0
27:         μ_{cluster→S_ij} ← 0
28:       else
29:         μ_{S_ij→cluster} ← −∞
30:       end if
31:     end for
32:   end for
33:   // Execute message passing algorithm
34:   currentIteration ← 0
35:   bestObjective ← −∞
36:   solution ← null
37:   repeat
38:     for each edge(i − j) in graph.edges() do
39:       edge.passMessage(computeMessage(i → j))
40:       objective ← computeObjective(graph)
41:       if objective > bestObjective then
42:         bestObjective ← objective
43:         solution ← decodeSolution(graph)
44:       end if
45:     end for
46:     currentIteration ← currentIteration + 1
47:   until (calibrationError(graph) ≤ MIN_ERROR
          or currentIteration ≥ MAX_ITERATIONS
          or objective has been decreasing for
              MAX_DIVERGENT_ITERATIONS)
48:   return solution
49: end procedure
50: procedure decodeSolution(graph)
51:   for each cluster_{S_ij} in graph.univarClusters() do
52:     val_ij ← cluster_{S_ij}.valueWithMaxBelief()
53:   end for
54:   return {val_ij}
55: end procedure
```

IV. Evaluation

In this section, we analyze the performance of our reaction execution engine and demonstrate how our proposed approach can be applied to a real-life problem, where we use the multi-agent constrained task assignment problem as a case study.

A. Performance Analysis

To evaluate the performance of the message-passing algorithm, we have performed over $2,500$ runs using the input parameters shown in Table I, whose values were selected randomly within the indicated ranges.

TABLE I. Minimum and maximum limits used for generating random input parameters for each experiment.

Parameter	Min	Max
Num. unique reactant types	100	1,000
Num. instances of each reactant type	1	5
Num. reaction rules	1	30
Num. unique reactant types in each reaction input multiset	1	5
Reactant type multiplicity in reaction input multiset	1	# reactant instances

Depending on the parameter values generated for each run, a different number of *assignable* h_{ij} variables (henceforth referred to as *matches*) end up in the cluster graph. As mentioned in Section III-D2, this number is often much smaller than the total number of variables in the graph, and is a better estimate of the true size of the optimization problem. Therefore, it is used as a key metric in this analysis. The algorithm was implemented in Java, and all experiments were executed on the Java HotSpot^{TM} 64-bit JVM v1.8.0_25-b17 running on a 2.4 GHz Intel Core i5 computer with 8 GB 1067 MHz DDR3 RAM.

Figure 5 shows a scatter plot comparing the actual objective and number of matches obtained using our

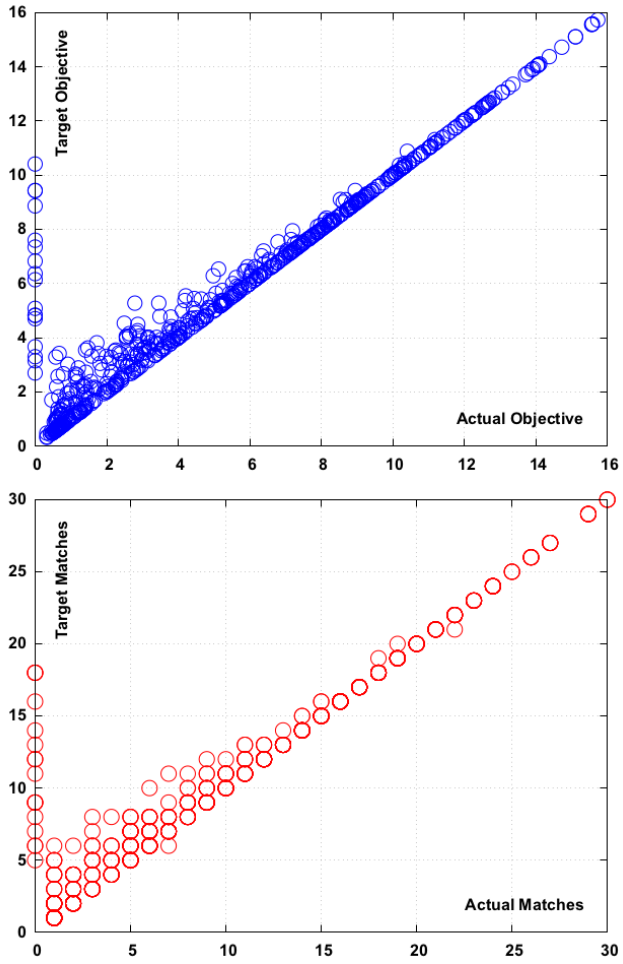

Figure 5. Scatter plot for Actual vs. Target objective (top), and number of matches (bottom).

algorithm to the same metrics when computed using a brute-force algorithm that finds the best solution using exhaustive search. The objective is computed using equation (2), and it represents the total reward gained by the system given a particular set of matches. Due to the intractability of computing the optimal solution, we have restricted comparisons with the brute-force algorithm to problems with 30 or less possible matches (which would require evaluating over 1 billion different combinations). The results in Figure 5 were obtained from over 600 runs, and they show that our algorithm matches or closely approaches the optimal solution in the majority of cases.

Figure 6 displays a break-down of the results in Figure 5. The histogram shows that our algorithm achieved 90% or more of the quality of the optimal objective in almost 77% of runs. The second largest group represents just 8% of runs, where the obtained objective lied between 50% and 70% of the optimal. In almost 2% of runs, our algorithm did not converge to any valid solution. Those runs are represented by the data points lying on the Y-axis in Figure 5. One possible circumvention in this case is to greedily select the non-conflicting matches that maximize the objective. This would usually yield an inferior solution, especially in the presence of many conflicts, but would still be preferable to selecting no reactions at all.

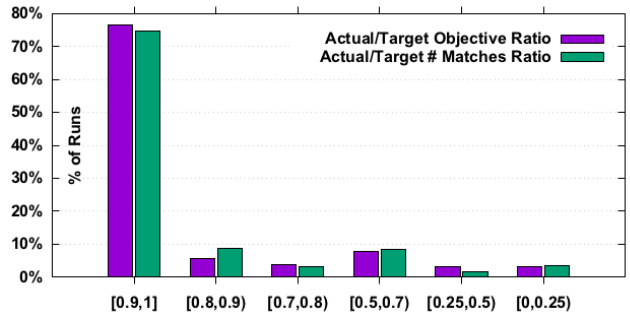

Figure 6. A break-down of the percentage of runs vs. actual/optimal solution ratio.

We now turn our attention to time performance. Figure 7 shows the number of message-passing iterations as well as the average number of messages/cluster that were needed to arrive at the solution. By combining the largest two groups in the histogram, we find that in more than 80% of runs, the best solution was found in less than 10 message-passing iterations and less than 30 message exchanges with every cluster in the graph.

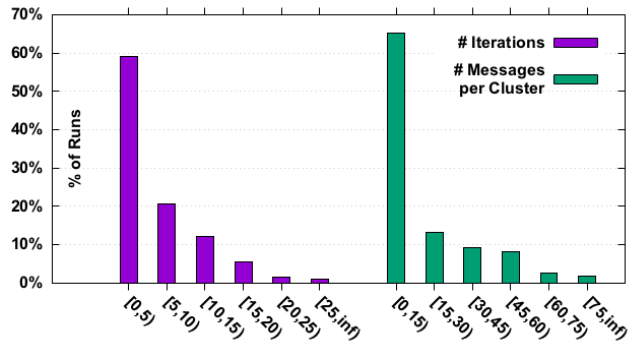

Figure 7. A break-down of the percentage of runs vs. the number of iterations and messages per cluster required to find the solution.

Figure 8 shows a comparison between the execution times of three different algorithms versus the number of possible matches in the cluster graph. In addition to the brute-force algorithm mentioned earlier, we have also added a greedy algorithm to act as a baseline for time performance. The greedy algorithm sorts all matches descendingly by reward then iterates over the sorted matches attempting the two possible assignments of each match and keeping the one that maximizes the sum of rewards.

Unsurprisingly, the message-passing algorithm is slower than the $O(N)$ greedy algorithm, but as can be seen in Figure 8, the execution times of both algorithms remain to be within a constant factor of each other, even when the number of possible matches exceeds 500. Figure 9 focuses on the message-passing algorithm, where each data point represents the average execution time of all runs that share the same number of matches.

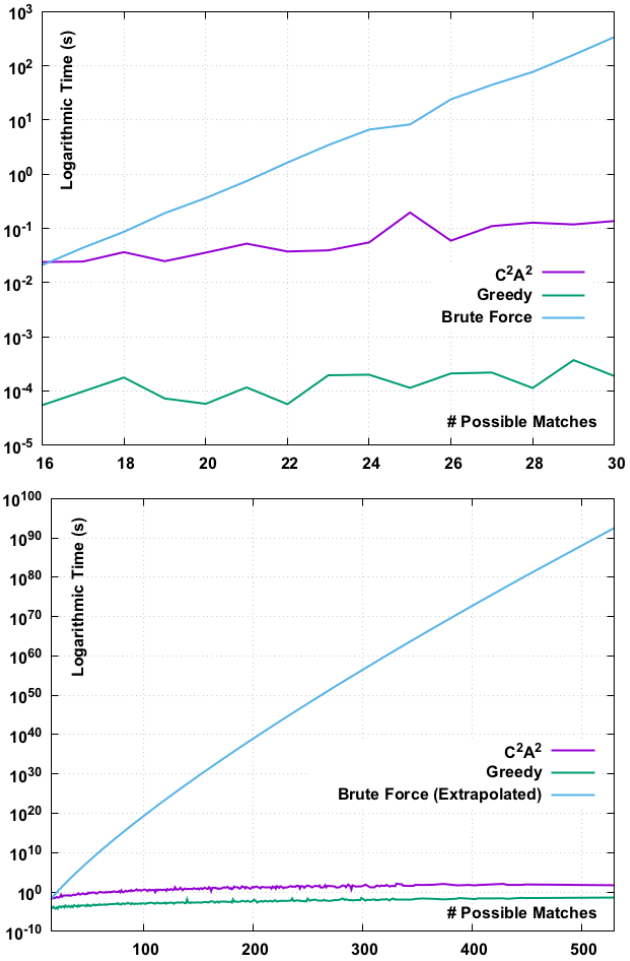

Figure 8. Logarithmic execution time vs. number of possible matches.

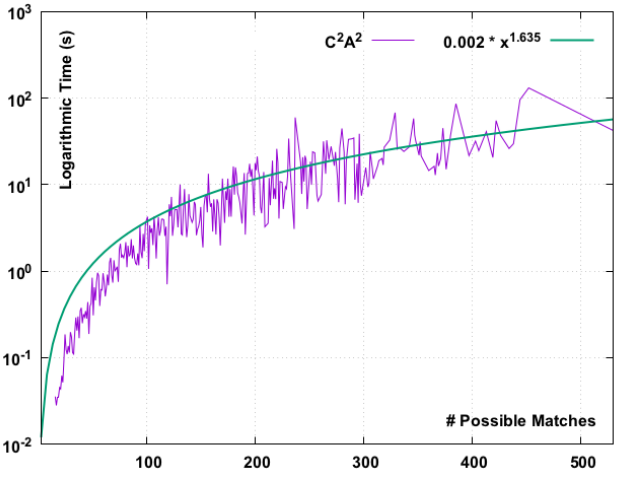

Figure 9. Execution time of message-passing algorithm vs. number of matches, showing the trend line of data points across all runs.

Finally, as validation of the quality of the solution obtained using the message-passing algorithm, we have compared it to the greedy solution as more reaction rules contend over the same reactant. In order to do this, we have generated sets of $1,000$ runs each. In every set, each reactant

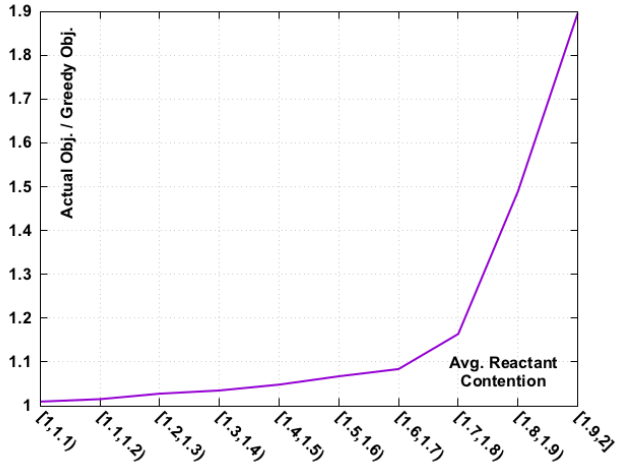

Figure 10. Reactant contention vs. improvement over greedy objective.

has an average number of reaction rules contending over it that falls within a predefined range. Figure 10 shows that the objective obtained using the message-passing algorithm improves exponentially relative to the greedy algorithm as the contention rate increases.

B. Case Study: Resource-Constrained Task Assignment

We now demonstrate how C_2A_2 can be applied to a concrete problem in a pervasive computing setting. The case study we have chosen is the problem of resource-constrained task assignment in a multi-agent system. Different versions of this problem have been studied in the literature, each with a slightly different problem statement (e.g., [23], [24], [25], [26]). The version we are interested in here has the following problem statement: given a set of tasks T and a set of heterogeneous agents A, where each task $t \in T$ is associated with a *demand* vector d_t the describes the requirements needed to starts the execution of task t, and each agent $a \in A$ is associated with a *supply* vector p_a that describes the capabilities of a, it is required to find an assignment $M : A \rightarrow T$ such that:

- For each task $t \in T$, M must contain an assignment associating t with a set of agents whose aggregate supply satisfies d_t.
- The assignment M should be selected such that the objective function $f = \sum_{t \in T} w(t) - \sum_{m \in M} c(m_{a \rightarrow t})$ is maximized, where $w(t)$ is the reward gained by the system for executing task t, and $c(m_{a \rightarrow t})$ is the cost incurred by the system for assigning agent a to task t.

[27] shows that this problem is $\mathcal{NP}$-hard, and therefore finding the optimal solution is prohibitively expensive for large inputs. Several centralized and distributed heuristic algorithms have been proposed in [23], [24], [25], [26] among others. In [24], an online centralized algorithm is proposed for forming coalitions among mobile heterogeneous robots, allowing them to perform tasks that require the collaboration of multiple robots with different capabilities. A test scenario that involves 10 robots and 3 tasks is given, for which we are now going to devise a solution using C_2A_2.

The robot capabilities required in the test scenario are shown in Table II, along with baseline values for the various functional elements specific to each capability. The scenario consists of three tasks, where each task imposes two levels of constraints: (a) robot-level constraints, which must be satisfied individually by each robot involved in executing the task, and (b) coalition-level constraints, which must be satisfied by the robot coalition as a whole. The three tasks are shown in Table III. Finally, the ten robots involved in the test scenario and the different capabilities they possess are shown in Table IV.

The scenario can be modeled using the reaction rules shown below. The number of rules correspond to the number of tasks, where for each task t, a corresponding rule is responsible for producing a robot coalition, c_t, with collective capabilities that satisfy the task requirements. In each rule, n is the number of robots in the coalition ($1 \leq n \leq 10$), and $|s_i|$ is the magnitude of robot capability i possessed either by an individual robot or the whole coalition.

$$t_1 + \left(n \times r_{|s_4|\geq 1\times\{0.8,0.8,0.5\}}\right)_{\substack{|s_1|\geq 1\times\{0.5\}\\ |s_3|\geq 1\times\{0.5\}\\ |s_4|\geq 2\times\{0.8,0.8,0.5\}}} \longrightarrow c_1 \quad (13)$$

$$t_2 + \left(n \times r_{\substack{|s_4|\geq 1\times\{0.8,0.8,0.5\}\\ |s_6|\geq 1\times\{0.8,1\}}}\right)_{|s_4|\geq 2\times\{0.8,0.8,0.5\}} \longrightarrow c_2 \quad (14)$$

$$t_3 + \left(n \times r_{|s_5|\geq 12\times\{1,1\}}\right)_{\substack{|s_2|\geq 1\times\{0.5\}\\ |s_3|\geq 2\times\{0.5\}\\ |s_4|\geq 2\times\{0.8,0.8,0.5\}}} \longrightarrow c_3 \quad (15)$$

Just like in [24], we are assuming that tasks are independent of each other, but the same concept can still be applied to dependent tasks, where a scheduler would be responsible for detecting which tasks may compete for the available resources at any given time and add the necessary reaction rules for them. The sequential scheme proposed in [24] uses a greedy heuristic to select the robot coalitions, but we will show that our concurrent solution can achieve a conflict-free assignment for all tasks with better resource utilization.

The next step is to specify the different affinity formulae that govern reaction execution, and consequently coalition formation. We consider the optimal solution to be the one that achieves the least possible surplus of unutilized capabilities, thus freeing up more robots to participate in any new tasks that may arise. Accordingly, we are going to construct the affinity formulae such that they optimize this objective. However, it can easily be shown that different definitions of optimality can be catered for by designing different affinity formulae.

We define the following affinity relationships in our solution:

- ***task-to-rule*** **affinity**: Because each rule in equations (13)-(15) is designed to produce a coalition for a particular task, the affinity between a task t and rule l is defined such that $aff(t,l) = -\infty$ unless l produced coalition c_t:

$$aff(t,l) = \begin{cases} 0, & \text{if output}(l) = c_t \\ -\infty, & \text{otherwise} \end{cases} \quad (16)$$

- ***robot-to-robot*** **affinity**: In this particular test scenario robots have no preference as to which other robots may join them in a coalition. Therefore, affinities between all pairs of robots are set to 0.
- ***robot-to-task*** **affinity**: The affinity between robot r and task t is computed according to equation (17). If r does not satisfy the robot-level constraints of t, then r cannot join coalition c_t, and thus $aff(r,t)$ is set to $-\infty$. On the other hand, if r meets the robot-level constraints of t, then $aff(r,t)$ decreases as the average capability surplus of r increases beyond the requirements of task t.

$$aff(r,t) = \begin{cases} -avg_{i:|s_i^{(r)}| \geq |s_i^{(t)}|} \{s_i^{(r)} - s_i^{(t)}\}, & \text{if } r \equiv t \\ -\infty, & \text{otherwise} \end{cases} \quad (17)$$

- ***coalition-to-rule*** **affinity**: For every rule in equations (13)-(15), C_2A_2 has to generate all possible input reactant groups that satisfy the rule (as discussed in more detail in Section III-B). In our test scenario, all of these groups are comprised from the task associated with the rule in addition to a candidate robot coalition that is compatible with the task. As such, the affinity between a candidate coalition cc_t and a rule l is defined as the average robot-to-task affinity (equation (17)) over all robots in the coalition if output(l) = t, and $-\infty$ otherwise:

$$aff(cc_t,l) = \begin{cases} avg_{r\in cc_t} \{aff(r,t)\}, & \text{if output}(l) = c_t \\ -\infty, & \text{otherwise} \end{cases} \quad (18)$$

Having defined the reaction rules and affinity formulae, we can now feed them into the reaction execution engine to obtain the solution. This particular problem has 608 conflict-free solutions, out of which only 55 (~9%) satisfy the resources needed by all three tasks. Figure 11 shows the possible objective values achievable by those 55 solutions and the number of occurrences of each objective. The results show that C_2A_2 was able to converge to the second best solution, which is compared to the optimal solution in Table V. The table also shows the magnitude of unutilized resources on robots assigned by each solution to the three tasks.

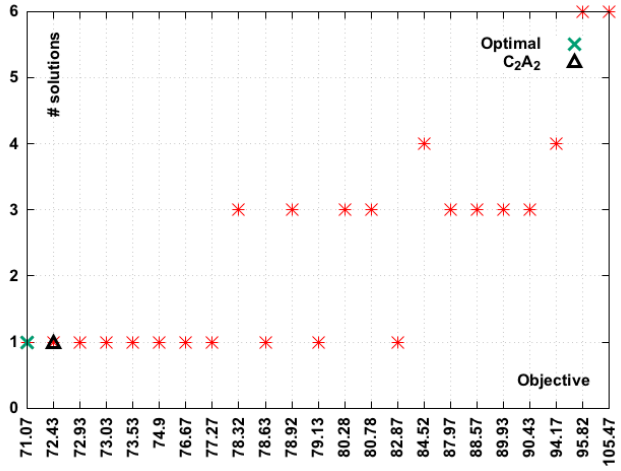

Figure 11. Number of conflict-free solutions for each objective value.

TABLE II. Robot capabilities used in test scenario, with baseline values for the functional elements specific to each capability.

Symbol	Description	Attributes
s_1	robot arm	$w_{lift} = 10kg$ (max. lifting capacity)
s_2	gripper	$w_{grip} = 20kg$ (max. object weight)
s_3	container	$w_{carry} = 50kg$ (max. payload)
s_4	camera	$a_{view} = 90°$ (viewing angle) $d_{view} = 5m$ (visible range) $f_{sampling} = 5frame/s$ (sampling frequency)
s_5	sonar	$d_{det} = 5m$ (sensor range) $f_{det} = 40Hz$ (scan frequency)
s_6	mobility	$v_{max} = 0.6m/s$ (max. linear velocity) $\omega_{max} = 0.5\pi/s$ (max. angular velocity)

TABLE III. Tasks used in the test scenario and their robot-level and coalition-level constraints. The magnitude of each capability functional element is mapped to the range $[0, 1]$, with 1 being the baseline value shown in Table II.

Symbol	Description	Robot-level Constraints	Coalition-level Constraints
t_1	area search and object manipulation	$1 \times s_4$ (camera) $\{0.8, 0.8, 0.5\}$	$1 \times s_1$ (robot arm) $\{0.5\}$ $1 \times s_3$ (container) $\{0.5\}$ $2 \times s_4$ (camera) $\{0.8, 0.8, 0.5\}$
t_2	area exploration	$1 \times s_4$ (camera) $\{0.8, 0.8, 0.5\}$ $1 \times s_6$ (mobility) $\{0.8, 1\}$	$2 \times s_4$ (camera) $\{0.8, 0.8, 1\}$
t_3	cargo transportation	$12 \times s_5$ (sonar) $\{1, 1\}$	$1 \times s_2$ (gripper) $\{0.5\}$ $2 \times s_3$ (container) $\{0.5\}$ $2 \times s_4$ (camera) $\{0.8, 0.8, 0.5\}$

TABLE IV. Capability matrix of robots used in the test scenario.

Capability/ Robot	s_1 (robot arm)	s_2 (gripper)	s_3 (container)	s_4 (camera)	s_5 (sonar)	s_6 (mobility)
r_1	$1 \times \{0.5\}$		$1 \times \{0.5\}$	$1 \times \{0.8, 0.8, 0.5\}$	$16 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_2		$1 \times \{0.6\}$		$1 \times \{0.8, 0.8, 0.5\}$	$16 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_3	$1 \times \{0.5\}$		$1 \times \{1\}$		$16 \times \{1, 1\}$	$1 \times \{0.5, 0.8\}$
r_4				$1 \times \{0.8, 0.8, 0.5\}$	$16 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_5				$1 \times \{0.8, 0.8, 1\}$	$14 \times \{1, 1\}$	$1 \times \{1, 1\}$
r_6		$1 \times \{1\}$	$1 \times \{1\}$		$14 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_7				$1 \times \{1, 1, 0.5\}$	$14 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_8	$1 \times \{0.5\}$			$1 \times \{1, 1, 0.5\}$	$14 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_9		$1 \times \{0.3\}$		$1 \times \{0.8, 0.8, 1\}$	$14 \times \{1, 1\}$	$1 \times \{0.8, 1\}$
r_{10}	$1 \times \{1\}$			$1 \times \{0.8, 0.8, 0.5\}$	$8 \times \{1, 1\}$	$1 \times \{0.5, 0.8\}$

TABLE V. C_2A_2 vs. optimal task-to-robot assignment.

	C_2A_2	Optimal Solution
t_1	$\{r_1, r_{10}\}$	$\{r_1, r_{10}\}$
t_2	$\{r_5, r_9\}$	$\{r_5, r_9\}$
t_3	$\{r_3, r_4, r_6, r_7\}$	$\{r_3, r_6, r_7, r_8\}$
	Resource Surplus	
s_1	$1 \times \{1\}$ $1 \times \{0.5\}$	$1 \times \{1\}$ $2 \times \{0.5\}$
s_2	$1 \times \{0.5\}$ $1 \times \{0.3\}$	$1 \times \{0.1\}$ $1 \times \{0.5\}$
s_3	$2 \times \{0.5\}$	$2 \times \{0.5\}$
s_4	$1 \times \{0.2, 0.2, 0\}$	$2 \times \{0.2, 0.2, 0\}$
s_5	$64 \times \{1, 1\}$	$62 \times \{1, 1\}$
s_6	$4 \times \{0.8, 1\}$ $2 \times \{0.5, 0, 8\}$ $1 \times \{0.2, 0\}$	$4 \times \{0.8, 1\}$ $2 \times \{0.5, 0, 8\}$ $1 \times \{0.2, 0\}$
Objective	72.43	71.07

V. Conclusion and Future Work

In this paper we presented a novel implementation of a chemical reaction execution engine geared toward multi-mission pervasive computing applications. By utilizing the concept of chemical affinity between network nodes, or between nodes and certain tasks or missions, we demonstrated an approach by which the network as a whole can be steered toward convergence on a set of common goals through the distributed exchange of affinity values expressed on an individual basis. We showed how the problem can be represented as a Maximum-a-Posteriori assignment problem and how it can be solved efficiently using the Max-Product Belief Propagation algorithm after carefully constructing a probabilistic factor graph and designing the message update formulae for the problem. We validated the efficacy of our approach using simulation, and also more tangibly by applying it to the problem of resource-constrained task assignment.

As an ongoing effort to extend the work presented here, we are developing a formal framework for modeling various pervasive computing and Internet-of-Things applications, which will be responsible for abstracting the different physical and logical elements of the network as entities in the chemical domain. Furthermore, we are studying the effect of context on inter-reactant affinities, allowing the development of context-aware systems that can quickly adapt to their surroundings. Finally, we plan to utilize reinforcement learning techniques and exploratory self-adaptation, where the system associates past decisions with their effect on performance, enabling the system to self-optimize in anticipation of potential events expected to take place in the future.

REFERENCES

[1] M. ElGammal and M. Eltoweissy, "Chemistry-inspired, Context-Aware, and Autonomic Management System for Networked Objects," in FUTURE COMPUTING, The Seventh International Conference on Future Computational Technologies and Applications, 2015, pp. 38–47.

[2] J. D. Park and A. Darwiche, "Solving MAP Exactly using Systematic Search," in UAI'03: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc, Aug. 2002.

[3] D. Koller and N. Friedman, Probabilistic Graphical Models: Principles and Techniques, ser. Adaptive Computation and Machine Learning. MIT Press, 2009.

[4] J. Banâtre, A. Coutant, and D. Le Metayer, "A Parallel Machine for Multiset Transformation and its Programming Style," Future Generation Computer Systems, vol. 4, no. 2, 1988, pp. 133–144.

[5] G. Berry and G. Boudol, "The Chemical Abstract Machine," Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '90., 1992.

[6] J. Bergstra and I. Bethke, "Molecular Dynamics," The Journal of Logic and Algebraic Programming, vol. 51, no. 2, 2002, p. 193–214.

[7] G. Păun, "Membrane Computing," Fundamentals of Computation Theory, 2003.

[8] M. Viroli and M. Casadei, "Biochemical Tuple Spaces for Self-organising Coordination," in Proceedings of the 11th International Conference on Coordination Languages and Models (COORDINATION 2009), 2009, pp. 143–162.

[9] P. Kreyssig and P. Dittrich, "On the Future of Chemistry-Inspired Computing," Organic Computing — A Paradigm Shift for Complex Systems Autonomic Systems, vol. 1, 2011, pp. 583–585.

[10] V. Rajcsányi and Z. Németh, "The Chemical Machine: An Interpreter for the Higher Order Chemical Language," Euro-Par 2011: Parallel Processing Workshops, 2012.

[11] C. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, vol. 19, no. 1, 1982.

[12] D. T. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," The journal of physical chemistry, 1977.

[13] M. A. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," The Journal of Physical Chemistry, vol. 104, no. 9, 2000, pp. 1876–1889.

[14] A. Slepoy, A. P. Thompson, and S. J. Plimpton, "A Constant-time Kinetic Monte Carlo Algorithm for Simulation of Large Biochemical Reaction Networks," The Journal of Chemical Physics, vol. 128, 2008.

[15] D. Pianini, S. Montagna, and M. Viroli, "Chemical-oriented Simulation of Computational Systems with Alchemist," Journal of Simulation, vol. 7, 2013, pp. 202–215.

[16] M. Monti, L. Sanguinetti, C. F. Tschudin, and M. Luise, "A Chemistry-Inspired Framework for Achieving Consensus in Wireless Sensor Networks," Sensors Journal, IEEE, vol. 14, no. 2, 2014, pp. 371–382.

[17] T. Meyer, "On Chemical and Self-healing Networking Protocols," Ph.D. dissertation, PhD Thesis, University of Basel, 2010.

[18] M. Monti, T. Meyer, C. F. Tschudin, and M. Luise, "Stability and Sensitivity Analysis of Traffic-Shaping Algorithms Inspired by Chemical Engineering," IEEE Journal on Selected Areas in Communications, vol. 31, no. 6, 2013, pp. 1105–1114.

[19] J. Pearl, "Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach," in AAAI, 1982, pp. 133–136.

[20] J. Banâtre, P. Fradet, and Y. Radenac, "Generalised Multisets for Chemical Programming," Mathematical Structures in Computer Science, Cambridge University Press, vol. 16, no. 04, 2006, pp. 557–580.

[21] D. Dueck, "Affinity Propagation: Clustering Data by Passing Messages," Citeseer, 2009.

[22] I. Givoni, "Beyond Affinity Propagation: Message Passing Algorithms for Clustering," Ph.D. dissertation, tspace.library.utoronto.ca, University of Toronto, 2012.

[23] S. S. Ponda, L. B. Johnson, and J. P. How, "Distributed chance-constrained task allocation for autonomous multi-agent teams," 2012 American Control Conference - ACC 2012, 2012, pp. 4528–4533.

[24] J. Chen and D. Sun, "Resource constrained multirobot task allocation based on leader–follower coalition methodology," The International Journal of Robotics Research, vol. 30, no. 12, Oct. 2011, pp. 1423–1434.

[25] T. Mercker, D. W. Casbeer, P. T. Millet, and M. R. Akella, "An Extension of Consensus-Based Auction Algorithms for Decentralized, Time-Constrained Task Assignment," 2010 American Control Conference (ACC 2010), 2010, pp. 6324–6329.

[26] S. Xu, "Energy-efficient Task Assignment of Wireless Sensor Network with The Application to Agriculture," Iowa State University Digital Repository, 2010.

[27] C.-H. Fua and S. S. Ge, "COBOS: Cooperative Backoff Adaptive Scheme for Multirobot Task Allocation," IEEE Transactions on Robotics, vol. 21, no. 6, Dec. 2005, pp. 1168–1178.