Protocols for Efficient Inference Communication

Carl Andersen and Prithwish Basu
Raytheon BBN Technologies
Cambridge, MA
canderse@bbn.com pbasu@bbn.com

Basak Guler and Aylin Yener and Ebrahim Molavianjazi
The Pennsylvania State University
University Park, PA 16802
bxg215@psu.edu yener@ee.psu.edu eum20@psu.edu

***Abstract*—Semantic approaches are increasingly applied to large-scale computing tasks. Nodes in such systems will need to exchange large volumes of declarative information, including ontologies and logical theories. In this paper, we describe our initial work on a system for compressed two-way communication of simplified logical inferences. This new system simplifies protocols for semantic data compression from our prior work, reducing seemingly different operations to hypergraph partitioning. We also present a new protocol that achieves significant speedup, at the cost of reduced compression.**

I. Introduction

Semantic approaches are increasingly applied to large-scale computing tasks, including web services [1], the internet of things [2], distributed databases [3], and data integration [4]. Nodes in such systems will need to exchange large volumes of declarative information of increasing expressiveness. While current computing relies on XML and other exchange languages, future systems may well exchange logical theories, ontologies, inferences, and other semantic content. This increasing prevalence of semantic data and communications creates a need for novel compression techniques (analogous to existing data and packet compression methods) that exploit underlying semantic structure (i.e. meaning).

Prior work by some of us [5] applied information- and graph-theoretic techniques to create communication protocols for compressed two-way communication of simplified logical inferences. This theoretical work also proved worst case upper bounds for the number of bits required to communicate semantic conclusions using the protocols.

The present paper describes four research accomplishments. First, we simplify the compression methods of the original protocols, reducing these to hypergraph partitioning operations. Second, we implement the revised protocols and evaluate their runtime and compressive performance (our original paper achieved purely proof-theoretic results). Third, we formulate and implement a new protocol that has greatly improved runtime efficiency. Finally, we analyze the QoI responsiveness of our compression techniques, which balance competing concerns of information quality and available resources.

II. The Scenario: Semantic Communication of Conclusions

The present paper retains the basic communication scenario of [5], as follows. Consider a network with two communicating agents A and B, who have shared knowledge of a semantic "theory" (set of statement) with a bipartite graph structure

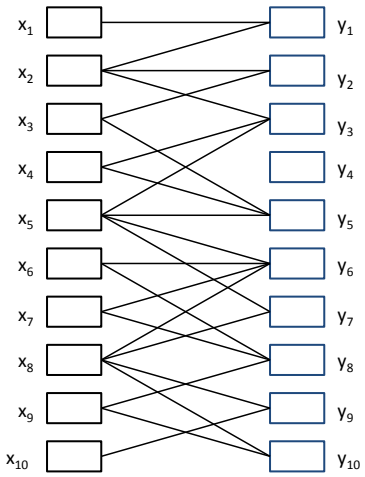

Fig. 1: Network model with 2 users and 10 facts for each user. Each line between facts is a conclusion; for greater visual clarity, these are unlabeled.

consisting of fact pairs linked to inferences, as depicted in Figure 1. While our statements are similar to propositional logic statements in form ($x \wedge y \rightarrow c$, meaning "if x and y are true, then so is c"), our theories do not implement most of the semantics of propositional logic, such as inferring contrapositives (as in "if c is false, then one of x or y must also be false").

More formally, the model contains two sets of facts, $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$ and $\mathcal{Y} = \{y_1, \ldots, y_{|\mathcal{Y}|}\}$, and one set of inferences or conclusions $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$. For our simplified model, we will assume that $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{C}$ are all disjoint. Each conclusion c_k is implied by exactly one pair of fact antecedents x_i and y_j drawn from $\mathcal{X}$ and $\mathcal{Y}$, respectively; we represent the inference relationship as $x_i \wedge y_j \rightarrow c_k$.

Our communication scenario is a semantic version of the classical information theory problem of communication with side information. We assume that the agents have shared knowledge of the structure of the theory graph. One conclusion $\mathring{c} \in \mathcal{C}$ with associated antecedents $\mathring{x} \in \mathcal{X}$ and $\mathring{y} \in \mathcal{Y}$ is chosen non-deterministically; agent A knows that $\mathring{x}$ is true, while agent B knows that $\mathring{y}$ is true. Each agent wants to learn $\mathring{c}$ and (equivalently) the other agent's antecedent. To achieve this shared understanding, the agents exchange communicated bit strings using a pre-arranged coding scheme that, ideally, minimizes the total bits sent.

Communication proceeds in rounds: in each round k, the

first agent communicates its bit string, ϕ_k^X, which the second agent may consider before replying with its bit string, ϕ_k^Y. A naive solution to this problem would involve a simple one-round exchange of binary codes representing the indices of the two antecedents, resulting in bit string lengths $|\phi_k^X| = \lceil \log(|\mathcal{X}|) \rceil$ and $|\phi_k^Y| = \lceil \log(|\mathcal{Y}|) \rceil$, respectively[1]. Our system uses a more subtle encoding scheme to achieve substantially smaller bit string lengths.

III. Additional Definitions: Hypergraphs, Partitions and Colorings

We first offer additional necessary concepts and definitions, some borrowed from [5]. Given bipartite graph $G = \langle \mathcal{X}, \mathcal{Y}, \mathcal{C} \rangle$, we define the *ambiguity set* $\mathcal{A}_i^X$ as the set of all possible facts from the second person that lead to a conclusion with x_i:

$$\mathcal{A}_i^X = \{y_j : x_i \wedge y_j \rightarrow c_k,\ y_j \in \mathcal{Y},\ c_k \in \mathcal{C}\} \tag{1}$$

Similarly, define the ambiguity set $\mathcal{A}_j^Y$ for every fact y_j as:

$$\mathcal{A}_j^Y = \{x_i : x_i \wedge y_j \rightarrow c_k,\ x_i \in \mathcal{X},\ c_k \in \mathcal{C}\} \tag{2}$$

We can group all the ambiguity sets A_i^X into an ambiguity collection A^X; A^Y is formed analogously.

Given bipartite graph $G = \langle \mathcal{X}, \mathcal{Y}, \mathcal{C} \rangle$, let $H_G^{\mathcal{X}} = (V, E)$ be the hypergraph having a vertex set $V = \mathcal{X}$ and an edge set E that is precisely the ambiguity collection $\mathcal{A}^{\mathcal{Y}}$. In other words, each $y \in \mathcal{Y}$ defines a hyperedge $e \in E$ such that $e = \{x : x \wedge y \rightarrow c,\ x \in \mathcal{X},\ c \in \mathcal{C}\}$. We define $H_G^{\mathcal{Y}}$ analogously.

A hypergraph partitioning is a division of the hypergraph's vertex set into disjoint subsets. Formally, given hypergraph $H_G^{\mathcal{X}} = (\mathcal{X}, \mathcal{A}^{\mathcal{Y}})$, we define a *hypergraph partitioning* $P_G^{\mathcal{X}} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots \mathcal{X}_n\}$ of $\mathcal{X}$, and call each $\mathcal{X}_i$ a *partition*. We define a partitioning $P_G^{\mathcal{Y}}$ analogously. Sample partitionings $P_G^{\mathcal{X}}$ and $P_G^{\mathcal{Y}}$ are shown in Figure 2.

Typically, we will not compute a hypergraph partitioning arbitrarily, but will instead choose a partition that has special properties useful for compressed communication. Consider a partitioning $P_G^{\mathcal{X}} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots \mathcal{X}_n\}$ of $\mathcal{X}$ having the property $|\mathcal{X}_k \cap e| \leq o$, for all $1 \leq k \leq n$ and $e \in \mathcal{A}^{\mathcal{Y}}$. We say that this partitioning has a *maximum overlap* between any partition and any hyperedge of o. If the maximum overlap $o = 1$, we say that the partitioning $P_G^{\mathcal{X}}$ is a *coloring* of $H_G^{\mathcal{X}}$, and we may refer to a given partition $\mathcal{X}_k$ as a *color*, because the partitions can be thought of as a mathematical coloring in which any two members of a given $\mathcal{A}^{\mathcal{Y}}$ must have different colors.

IV. Semantic Coding Via Hypergraph Partitioning

In this section, we describe compressed communication techniques that rely on hypergraph partitioning. In general, our protocol uses hypergraph partitioning upon $H_G^{\mathcal{X}}$ to divide $\mathcal{X}$ into a small number of disjoint subsets. Then, instead of communicating which fact $x \in \mathcal{X}$ it posseses, an agent communicates which partition x is in. Each agent can combine the knowledge of its own fact and the received partition to make further inferences, narrowing down the possibilities for the other agent's fact. After a finite number of rounds of this kind of communication, each agent is able to infer the other's fact.

[1]Throughout the paper, we assume that agents can encode which member of some n-ary set is present using a bit string of length $\log(n)$. Ideally, to avoid ambiguity, a communication system would actually use a prefix coding system such as Huffman coding. Here we opted for simplicity of both explanation and implementation.

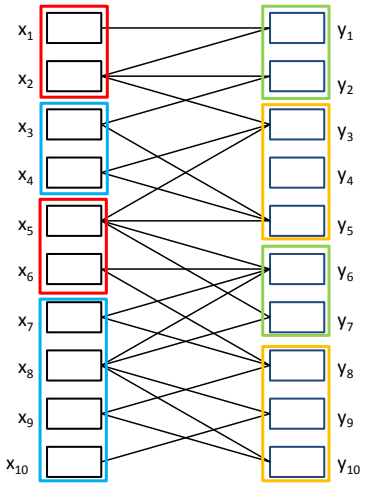

Fig. 2: Partitionings $P_G^{\mathcal{X}}$ and $P_G^{\mathcal{Y}}$ for the graph G from Figure 1. $\mathcal{X}$ is partitioned into two partitions, denoted here by the red and blue boxes. $\mathcal{Y}$ is also partitioned into two partitions, denoted here by the green and orange boxes. Note that despite our use of colored boxes, neither partitioning is a *coloring*. Each of the two partitions has a maximum overlap $o = 2$ with any ambiguity set, i.e. with the edges defined by any one vertex on the other side of the graph.

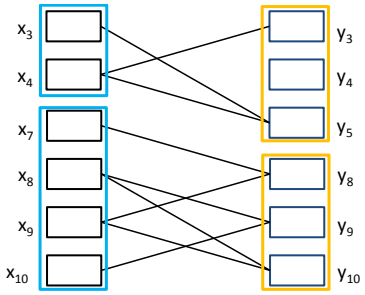

Fig. 3: A restricted graph formed from the blue and orange partitions from Figure 2.

A. Candidate Reduction Using Hypergraph Partitioning

We already assume that both sides share knowledge of the bipartite graph G; they therefore can both derive the hypergraph $H_G^{\mathcal{X}} = (\mathcal{X}, \mathcal{A}^{\mathcal{Y}})$. Now, consider a partitioning $P_G^{\mathcal{X}} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots \mathcal{X}_n\}$ of $\mathcal{X}$ having maximum overlap of o. Suppose both sides use the same (pre-arranged) partitioning algorithm to independently obtain $P_G^{\mathcal{X}}$. Now suppose A communicates to B that its fact is in partition $\mathcal{X}_k$, using $\log(n)$ bits. At this point, B, who knows its fact, $\mathring{y}$, has index j, can infer that A's fact $\mathring{x}$ is one of the o facts in $\mathcal{X}_k \cap A_j^Y$. In the case that we computed the partition set so that $o = 1$, B

can infer A's fact exactly. Even if $o > 1$, the partition step and associated communication has enabled B to significantly reduce the number of possibilities for A's fact.

B. Restricted Graphs

If a partition has maximum overlap $o > 1$, it can still be used to simplify the graph, as follows. Suppose the agents each compute partitionings $P_G^{\mathcal{X}}$ and $P_G^{\mathcal{Y}}$, again using a prearranged partitioning algorithm. The agents then exchange the respective partitions that each's fact resides in; let these partition indices be i and j, respectively. Now each agent knows that $\mathring{x}$ and $\mathring{y}$ are restricted to the facts in the two partitions $\mathcal{X}_i$ and $\mathcal{Y}_j$, respectively. The agents can then independently construct a new graph which restricts the vertices to those found in the partitions. Formally, the agents each construct the *restricted graph* $G' = \langle \mathcal{X}', \mathcal{Y}', \mathcal{C}' \rangle$, where $\mathcal{X}' = \{x \in \mathcal{X}_i\}$ and $\mathcal{Y}' = \{y \in \mathcal{Y}_j\}$ and $\mathcal{C}' = \{c \in \mathcal{C} : x \in \mathcal{X}' \text{ and } y \in \mathcal{Y}' \text{ and } x \wedge y \rightarrow c\}$. We expect that the restricted graph will be significantly smaller than the original graph, so it will be easier to partition and also not require as many bits to encode. The agents can therefore continue their communications over further rounds, this time using the restricted graph for greater efficiency.

C. Meta-graphs

A different formalism that supports successive rounds of partitioning plus communication is a *meta-graph*. Given a graph G, partitionings $P_G^{\mathcal{X}} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots \mathcal{X}_m\}$ and $P_G^{\mathcal{Y}} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots \mathcal{Y}_n\}$, we can define a meta-graph G' in which the partition indices themselves are the facts. Formally, $G' = \langle \mathcal{X}', \mathcal{Y}', \mathcal{C}' \rangle$, where $\mathcal{X}' = \{1, 2, ..., m\}$ and $\mathcal{Y}' = \{1, 2, ..., n\}$. We then define conclusions in the meta-graph as follows. For each pairing of $i \in \mathcal{X}'$ and $j \in \mathcal{Y}'$, the conclusion $\langle i, j \rangle$ is in $\mathcal{C}'$ iff exists some $x \in \mathcal{X}_i, y \in \mathcal{Y}_j, c \in \mathcal{C}$ such that $x \wedge y \rightarrow c$. Once the meta-graph G' is constructed, it is of course possible to construct $H_{G'}^{\mathcal{X}}$ and $H_{G'}^{\mathcal{Y}}$, and then partition these. Therefore, meta-graphs make it possible to represent repeated partitioning of the original graph. Note that unlike restricted graphs, the construction of meta-graphs requires no exchange of partitions or other information between the agents.

D. Existence of a Hypergraph Partitioning

It is not obvious that a partitioning as described above, with its associated number of partitions n and maximum overlap o, must exist for an arbitrary hypergraph. Our work is guided by the following result, also used in [5]:

Lemma 1. *[6] Let $H = (V, E)$ be a hypergraph with a vertex set V of size $|V|$. Hyperedges are given as $E_i \subseteq V$ for $i = 1, 2, \ldots, |E|$ and each hyperedge consists of at most d elements, i.e., $|E_i| \leq d$. Given $\epsilon > 0$, there exists a constant $c(\epsilon)$ such that $\forall p \geq (\ln \sqrt{|V||E|})^{1+\epsilon}$ and $p > 1$, a partitioning $V_1, V_2, \ldots V_{\lceil \frac{d}{p} c(\epsilon) \rceil}$ of V can be found with the property $|V_k \cap E_i| < p$, for $i = 1, \ldots, |E|$ and $k = 1, \ldots, \lceil \frac{d}{p} c(\epsilon) \rceil$.*

Note that Lemma 1 provides an existence proof, showing that a partitioning with the desired properties must exist, rather than offering a tractable algorithm for finding it. Our implementation of partitioning, described later, uses the lemma primarily as a guide for reasonable parameters to attempt in partitioning efforts, e.g. the number of partitions. Also note that the lemma uses a non-inclusive bound p for the overlap, so $p = o + 1$, where o is our term expressing the maximum inclusive overlap.

V. Communication Protocols

In this section, we describe three different communication protocols between A and B; each uses one or more communication rounds interleaved with the graph construction and partitioning operations described earlier. All graph construction and partitioning steps are performed independently but identically by both A and B, who are assumed to use exactly the same algorithms for construction and partitioning. Each protocol attempts to minimize the code lengths (number of bits required to be exchanged between A and B) while still guaranteeing that A and B know each other's fact by the end of the protocol. In Section VII, we conduct empirical tests of the code lengths and also of the runtime required by each protocol's associated graph construction and partitioning operations.

Protocols 1 and 2 are adaptations of protocols from our theoretical paper [5]; we simplify the original protocols by reducing all their key operations to hypergraph partitioning. Protocol 3 is new.

A. Protocol 1

In this protocol, A and B use hypergraph coloring operations to reduce the number of bits required to infer each other's fact. Our theoretical paper proved ([5], Theorem 2) an upper bound on code lengths for Protocol 1.

1) From the initial bipartite graph G_1, construct hypergraphs $H_{G_1}^{\mathcal{X}}$ and $H_{G_1}^{\mathcal{Y}}$.
2) Color each of $H_{G_1}^{\mathcal{X}}$ and $H_{G_1}^{\mathcal{Y}}$ (i.e., partition each with maximum overlap $o = 1$). For each partitioning problem, we use the equation from Lemma 1, with o fixed at 1, to determine n, the number of partitions required. In other words, if, e.g. $H_{G_1}^{\mathcal{X}}$'s largest hyperedge has size d, we search for a partitioning $P_G^{\mathcal{X}}$ having $\lceil \frac{d}{o} c(\epsilon) \rceil$ partitions. In this and other uses of Lemma 1, we use $\epsilon = .2$ and $c(\epsilon) = 1.2$.
3) (*Communication Round 1*) A and B exchange codes indicating the index of the partitions that their respective facts are in. This requires no more than $\log(m) + \log(n)$ bits, where m and n are the number of partitions used to partition $H_{G_1}^{\mathcal{X}}$ and $H_{G_1}^{\mathcal{Y}}$, respectively. We expect that m and n will be much smaller than $|\mathcal{X}|$ and $|\mathcal{Y}|$. Agent A can use its $\mathring{x}$ fact and its knowlege of B's partition to infer $\mathring{y}$. Agent B reasons analogously.

B. Protocol 2

In Protocol 1, the number of colors could still be very large if the graphs are complex. Protocol 2 attempts to achieve additional compression by creating a meta-graph from the

colors and partitioning it. The agents then exchange the codes for this second partition. A restricted graph technique is used to complete the inference chain, allowing each player to infer the other's color. Our theoretical paper proved ([5], Theorem 3) improved upper-bound code lengths over Protocol 1.

1) From the initial bipartite graph G_1, construct hypergraphs $H^{\mathcal{X}}_{G_1}$ and $H^{\mathcal{Y}}_{G_1}$.
2) (*Coloring*) Using the hypergraphs, compute partitionings $P^{\mathcal{X}}_{G_1}$ and $P^{\mathcal{Y}}_{G_1}$, with maximum overlap $o = 1$. [2]
3) Construct a meta-graph G_2 from the partitions $P^{\mathcal{X}}_{G_1}$ and $P^{\mathcal{Y}}_{G_1}$.
4) From G_2, construct hypergraphs $H^{\mathcal{X}}_{G_2}$ and $H^{\mathcal{Y}}_{G_2}$. [3] Partition $H^{\mathcal{X}}_{G_2}$ and $H^{\mathcal{Y}}_{G_2}$ with o and n from Lemma 1.
5) (*Communication Round 1*) A and B exchange codes indicating the index of the partitions from $P^{\mathcal{X}}_{G_2}$ and $P^{\mathcal{Y}}_{G_2}$, respectively, that their respective colors are in. These codes use $\log(m)$ and $\log(n)$ bits, respectively. Now each side knows the other's meta-graph partition, allowing each side to construct an identical restricted graph in the next step.
6) Construct a restricted graph G_3 from the two partitions $\mathcal{X}_i$ and $\mathcal{Y}_j$.
7) From G_3, construct hypergraphs $H^{\mathcal{X}}_{G_3}$ and $H^{\mathcal{Y}}_{G_3}$.
8) Partition $H^{\mathcal{X}}_{G_3}$ and $H^{\mathcal{Y}}_{G_3}$ with $o = 1$, n from Lemma 1.
9) (*Communication Round 2*) A and B exchange partition codes from G_3. Knowing its own partition from G_1, and the other's partition from G_3, each side can infer the other's partition from G_1.[4]
10) Using its fact and the other's partition from G_1, each side can infer the other's fact.

C. Protocol 3

We developed this protocol (inspired by a proof from [6]) because our testing indicated that finding a partition with a small maximum overlap o requires significantly more runtime than finding a partition with larger o. Protocols 1 and 2, which both begin with a hypergraph coloring (i.e. $o = 1$) are therefore very costly.

Protocol 3 pursues a more gradual approach, repeatedly constructing restricted graphs in response to user code exchanges. Each iteration prunes away some portion of the graph, until only $\mathring{x}$ and $\mathring{y}$ are left. Importantly, this protocol can be parameterized to engage in more or less aggressive partitioning by setting o. Setting o to higher values leads to less aggressive partitioning over a greater number of rounds. Because each round prunes away part of the graph, this strategy can often prune away complex subgraph components before they are ever partitioned with much rigor (i.e. using lower o). We found that o could be usefully varied between a low of o_{Lemma1}, the maximum overlap recommended by Lemma 1, and a high of $o_{\mathcal{A}^{\mathcal{X}}}$, the size of the largest hyperedge in $H^{\mathcal{X}}$. We explored a range of values of a factor ϕ, where $o = o_{Lemma1} + \phi(o_{\mathcal{A}^{\mathcal{X}}} - o_{Lemma1})$. We set the number of partitions using o and Lemma 1.

1) Let G_1 be the initial bipartite graph. While G_1 contains more than one x and y fact, iterate steps 2-5. After the last iteration, G_1 will contain only the two facts $\mathring{x}$ and $\mathring{y}$.
2) Construct hypergraphs $H^{\mathcal{X}}_{G_1}$ and $H^{\mathcal{Y}}_{G_1}$.
3) Using the hypergraphs, compute partitionings $P^{\mathcal{X}}_{G_1}$ and $P^{\mathcal{Y}}_{G_1}$, with maximum overlap o.
4) (*Communication Round*) A and B exchange partition codes from $P^{\mathcal{X}}_{G_1}$ and $P^{\mathcal{Y}}_{G_1}$, respectively, using $\log(m)$ and $\log(n)$ bits, respectively. Now each side knows the other's partition ($\mathcal{X}_i$ and $\mathcal{Y}_j$, respectively), allowing each side to construct an identical restricted graph in the next step.
5) Construct a restricted graph G_2 from the two partitions $\mathcal{X}_i$ and $\mathcal{Y}_j$. Set $G_1 := G_2$.

VI. Implementation

This section discusses our implementation of the three protocols and associated graph operations discussed above. Our implementation uses the Python programming language and associated packages. To build and manipulate graphs and hypergraphs, we use the *pygraph* package. While space prevents a detailed code presentation, we describe key functions used in the protocols briefly in Table I.

A. Partitioning Using Mixed Integer Programming

To find hypergraph partitionings, we expressed the partitioning problem as a Mixed Integer Programming (MIP) problem and solved them using the Python *pulp-or* package [7], which serves as a wrapper for numerous Operations Research packages implemented in other languages. We set *pulp-or* to call the *cbc* mixed integer programming solver, which supports a timeout function useful for our testing.

Mixed Integer Programming expresses optimization problems as sets of linear methematical formulas. Some of these formulas are constraints upon valid variable solutions, while others are objective functions that are maximized or minimized to find the most desired solutions. For increased speed, our MIP for finding a partition does not include an objective function. Our MIP determines 0 / 1 values for variables a_{vp} defined for each vertex $v \in V$ and partition $p \in P$; when $a_{vp} = 1$, v is held to be a member of p.

Mixed Integer Program 1. *partitioning($H^{\mathcal{X}}_G$,o,n): Given hypergraph $H^{\mathcal{X}}_G = (V, E)$, to discover a partitioning $P^{\mathcal{X}}_G = \{\mathcal{X}_1, \mathcal{X}_2, \ldots \mathcal{X}_n\}$ of $\mathcal{X}$ having maximum overlap o,*

$$\textit{find } a_{vp}, \qquad v \in V, p \in P^{\mathcal{X}}_G, \textit{each } a_{vp} = 0 \textit{ or } 1$$

$$\textit{subject to } \sum_{v \in e} a_{vp} \leq o \quad , \quad p \in P^{\mathcal{X}}_G, e \in E \quad \textit{(constraint 1)}$$

$$\sum_{p \in P^{\mathcal{X}}_G} a_{vp} = 1, \quad v \in V \qquad \textit{(constraint 2)}$$

[2] This step corresponds to the process detailed in our original paper [5] of constructing a characteristic graph and coloring it.

[3] This step corresponds to [5]'s construction of hypergraphs upon the set of colors.

[4] In [5], this step is broken into two steps, achieving a slightly better bound. For presentation reasons, this paper's version is simplified.

Function Name	Description
buildGraph(numX,numY,numConc)	Builds bipartite graph having numX and numY vertices, respectively, using Pygraph functions. A total of numConc conclusion edges are added randomly, discarding duplicates.
buildAmbiguitySet(graph,vert)	Builds an ambiguity set (i.e. a hyperedge) by finding y [x] neighbors of vert and constructing a hyperedge (i.e. a set) containing them.
buildHypergraph(graph,vertSet)	VertSet is either the X or Y set of vertices in graph. Builds a hypergraph by calling buildAmbiguitySet(graph,v) for each vertex v in vertSet.
partitioning(hypergraph,o,n)	Using MIP, finds a partitioning of hypergraph having max overlap o and number of partitions n. Outputs a set of partitions (each is a set of vertices).
buildRestricted-Graph(graph,partitionX,partitionY)	Constructs a new bipartite graph consisting of the vertices from partitionX and partitionY. The original graph is scanned and any edges between the vertices are added to the new graph. Any vertices not participating in an edge are deleted.
getPartition(othersCode,partitioning)	The player uses othersCode as an index into the partitioning (set of partitions) to retrieve a partition.
inferOthersVertex(graph,ownVertex,partition)	The player scans partition to find the unique vertex linked to his own vertex in graph.

TABLE I: Functions used in the implementation.

The above MIP assumes a pre-determined number of partitions n; we always use Lemma 1 to determine this number. The MIP also assumes a maximum overlap o; we always either fix o at 1 for coloring or compute it using Lemma 1 or ϕ.

VII. Evaluation

We ran tests upon all three protocols that measured their compressive power as well as the runtime required to achieve that compression.

A. Test Methodology

Our tests perform runs of protocols over random graphs generated using input numbers for $|\mathcal{X}|$, $|\mathcal{Y}|$ and $|\mathcal{C}|$. All our tests use $|\mathcal{X}| = |\mathcal{Y}|$. The random graphs are constructed by choosing pairs of $x \in \mathcal{X}$, $y \in \mathcal{Y}$ randomly and adding a conclusion $\langle x, y \rangle$ to the graph; duplicate conclusions are discarded and another conclusion generated to replace the duplicate.

We first investigated the scalability and compression of each protocol when $|\mathcal{X}|$, $|\mathcal{Y}|$ and $|\mathcal{C}|$ all increase at the same rate. Next, we investigated scalability and compression holding $|\mathcal{X}|$ and $|\mathcal{Y}|$ constant and increasing $|\mathcal{C}|$. For each tested combination of $|\mathcal{X}|$ and $|\mathcal{Y}|$ and $|\mathcal{C}|$, we generate five random graphs; for each of these, we perform five runs of the full protocol, each time choosing $\mathring{c}$ randomly. We validate each run by testing at the conclusion of the protocol that the agent's inferred fact actually matches the other agent's fact.

B. Results

Our test results, in Table II, show that for relatively large graph sizes, it is indeed possible to find a partition meeting the constraints of Lemma 1. However, the complexity of hypergraph partitioning is NP-hard [8], so we expect to see the time necessary to find partitions would be exponential in the number of hypergraph vertices and size of hypergraph edges. Our results reflect this exponential relationship.

Our tests show that all three protocols provide significant compression of the exchanged bits required to communicate a conclusion. Interestingly, our results from [5] proving the theoretical compressive superiority of Protocol 2 over Protocol 1 are not reflected in our tests. These empirical results do not actually conflict with our theoretical findings, but instead represent cases in which associated constants dominate the bit length equations. As graph and conclusion sizes increase further, we would expect to see the theoretical superiority of Protocol 2 assert itself. Unfortunately, because the partitioning operation is so expensive, we are unable to run tests of the size required to demonstrate this.

We also tested Protocol 3, which tries to break up the necessary partitioning needed to achieve compression into multiple, less computationally intensive partitioning steps. Our tests show that Protocol 3 is indeed far more scalable than either Protocols 1 or 2, requiring orders of magnitude less runtime to achieve nearly the same compression.

C. Results - QoI responsiveness

Interactive compression rates are directly related to Quality of Information (QoI) characteristics of semantic communications. Although achieving higher compression rates is desirable for delivering the same level of semantics, they come at the cost of higher runtime, and hence less QoI responsiveness. We attempted to establish a controlled relationship between the runtime devoted to the task and the resulting compression. This is difficult to achieve, because these protocols operate under stringent requirements. Aggressive partitioning in a small number of rounds can achieve significant compression, but also cause exponential runtime blowup. On the other hand, more relaxed partitioning (in which the maximum overlap o is high) can achieve little to no compression. Because each round of partitioning plus code exchange typically involves an exchange of 2-4 bits, multi-round schemes (like Protocol 3) have only a few rounds to prune the graph. Finally, often the transitions between over-relaxed and over-aggressive are abrupt.

We did achieve a measure of control by increasing the number of rounds and maximum overlap o used by the protocols. The columns of Table II show a general left-to-right

$\lvert\mathcal{X}\rvert$ / $\lvert\mathcal{Y}\rvert$ / $\lvert\mathcal{C}\rvert$	Naive	Protocol 1		Protocol 2		Protocol 3, base		Protocol 3, $\phi = 0.2$		Protocol 3, $\phi = 0.4$		Protocol 3, $\phi = 0.6$	
	bits	time	bits	time	bits	time	bits	time	bits	time	bits	time	bits
10 / 10 / 20	8	0.13	4	1.09	4	0.22	5	0.2	5	0.18	4	0.3	5
20 / 20 / 40	10	0.28	5	0.46	5	0.27	5	0.27	4	0.32	5	0.3	6
40 / 40 / 80	12	0.65	5	1.04	5	0.43	5	0.43	7	0.45	5	0.61	7
80 / 80 / 160	14	1.68	5	1.91	6	0.75	5	0.8	7	0.83	7	0.78	7
160 / 160 / 320	16	4.11	6	4.36	6	2.15	6	1.35	7	1.2	7	1.13	7
320 / 320 / 640	18	11.27	6	11.52	6	5.22	6	4.29	7	5.13	7	4.5	7
640 / 640 / 1280	20	37.95	6	38.24	7	18.78	7	8.5	7	8.7	7	10.56	8
640 / 640 / 2560	20	809.55	7	810.15	8	17.26	8	8.87	10	12.47	10	16.06	10
640 / 640 / 5120	20	timeout		timeout		32.04	9	11.09	11	8.82	10	13.29	12
640 / 640 / 10240	20	timeout		timeout		778.64	10	48.6	13	15.19	14	12.09	12
640 / 640 / 20480	20	timeout		timeout		timeout		354.73	14	19.84	15	15.48	16

TABLE II: Test results for Protocols 1-3. All time results are in seconds. The *timeout* value signifies runtime > 2000 seconds. The Naive Protocol simply encodes the exchanged facts using $\lceil\log(|\mathcal{X}|)\rceil + \lceil\log(|\mathcal{Y}|)\rceil$ bits. The Protocol 3 base run allows only one restricted graph iteration, after which the restricted graph is colored. The other runs of Protocol 3 iterate until they naturally achieve a coloring.

decrease in runtime coupled with a decrease in compression (i.e. the bits exchanged increase). This control is still irregular, reflecting our limited understanding of the forces driving partitioning complexity. In future work, we hope to leverage more sophisticated partitioning algorithms that allow finer, more accurate control over runtime.

VIII. Conclusions

This paper implemented and tested theoretical semantic compression protocols from a prior paper [5]. In addition, it reformulated a variety of operations from that paper as variations of repeated hypergraph partitioning. Finally, it offered a novel protocol that achieves close to the same compression as the earlier protocols in orders of magnitude less runtime. Another prior paper by one of us [9] explored theoretical approaches to the compression of propositional logic communication. However, we are not aware of any prior work implementing the semantic compression of communications in the presence of shared knowledge.

We plan future work along several dimensions. In addition to pursuing increased control over runtime (mentioned earler), we also will pursue improved scalability. Our current MIP implementation of partitioning is not particularly scalable, handling hypergraphs of thousands of edges with difficulty. We will adapt more mature implementations of hypergraph partitioning, such as [10], known to find partitions over graphs as large as millions of edges. We also plan to investigate alternative greedy methods for hypergraph coloring and methods that produce balanced-size partitions.

Also, our current model of inferential conclusions is limited: it does not reflect the true complexity of even propositional logic, in which conclusions can themselves be antecedents. In future work, we plan to extend our current methods and protocols to operate over propositional logic itself. One anticipated challenge of this extension is the requirement that agents communicate not just one fact, but possibly several, in order to enable a propositional inference. Another potential extension of our protocol would be to communicate rules in addition to facts.

References

[1] D. Fensel, F. M. Facca, E. Simperl, and I. Toma, *Semantic web services*. Springer, 2011.

[2] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, "Smart semantic middleware for the internet of things.," *ICINCO-ICSO*, vol. 8, pp. 169–178, 2008.

[3] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang, "A distributed graph engine for web scale rdf data," in *Proceedings of the VLDB Endowment*, VLDB Endowment, vol. 6, 2013, pp. 265–276.

[4] A. Cal`, D. Calvanese, G. De Giacomo, and M. Lenzerini, "Data integration under integrity constraints," in *Seminal Contributions to Information Systems Engineering*, Springer, 2013, pp. 335–352.

[5] B. Guler, A. Yener, and P. Basu, "A study of semantic data compression," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, IEEE, 2013, pp. 887–890.

[6] A. El Gamal and A. Orlitsky, "Interactive data compression," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, IEEE, 1984, pp. 100–108.

[7] S. Mitchell, M. O'Sullivan, and I. Dunning, "Pulp: a linear programming toolkit for python," *Sep-2011*, 2011.

[8] M. R. Garey and D. S. Johnson, *Computers and intractibility*, 1979.

[9] P. Basu, J. Bao, M. Dean, and J. Hendler, "Preserving quality of information by using semantic relationships," *Pervasive and Mobile Computing*, vol. 11, pp. 188–202, 2014.

[10] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Design Automation, 1982. 19th Conference on*, IEEE, 1982, pp. 175–181.