DCOPs and Bandits: Exploration and Exploitation in Decentralised Coordination

Ruben Stranders, Long Tran-Thanh, Francesco M. Delle Fave, Alex Rogers & Nicholas R. Jennings
University of Southampton
{rs2, ltt08r, fmdf08r,acr,nrj}@ecs.soton.ac.uk

ABSTRACT

Real life coordination problems are characterised by stochasticity and a lack of *a priori* knowledge about the interactions between agents. However, decentralised constraint optimisation problems (DCOPs), a widely adopted framework for modelling decentralised coordination tasks, assumes perfect knowledge of these factors, thus limiting its practical applicability. To address this shortcoming, we introduce the MAB–DCOP, in which the interactions between agents are modelled by multi-armed bandits (MABs). Unlike canonical DCOPs, a MAB–DCOP is not a single shot optimisation problem. Rather, it is a sequential one in which agents need to coordinate in order to strike a balance between acquiring knowledge about the *a priori* unknown and stochastic interactions (exploration), and taking the currently believed optimal joint action (exploitation), so as to maximise the cumulative global utility over a finite time horizon. We propose Heist, the first asymptotically optimal algorithm for coordination under stochasticity and lack of prior knowledge. Heist solves MAB–DCOPs in a decentralised fashion using a generalised distributive law (GDL) message passing phase to find the joint action with the highest upper confidence bound (UCB) on global utility. We demonstrate that Heist outperforms other state of the art techniques from the MAB and DCOP literature by up to 1.5 orders of magnitude on MAB–DCOPs in experimental settings.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Theory, Experimentation

Keywords

Coordination, Distributed Problem Solving, Uncertainty

1. INTRODUCTION

Many real life applications can be modelled as systems of coordinating autonomous agents. Examples include wireless sensor networks (WSN), teams of UAVs deployed in disaster response scenarios and scheduling multi-processor jobs with unknown duration in distributed computing environments. These application domains often require decentralised solution techniques, as these yield scalable solutions, exhibit rapid response times and are robust to failure by avoiding the existence of a centralised point of control. As a result, building effective and efficient coordination algorithms has been a focus of attention within the multi-agent community.

This focus has led to the development of the framework of the distributed constraint optimisation problem (DCOP) [13]. In a DCOP, the agents' objective is to maximise a global utility function, that can be factorised into a sum of local utility functions that represent the interactions between agents. However, the canonical DCOP framework makes two strong assumptions that limit its practical applicability. First, the local utility functions are assumed to be known *a priori*. Second, it is assumed that the local utility functions are non-stochastic, i.e. that the same interaction always yields the same outcome. In reality, not only are utility functions typically noisy, they are also unknown before the agents' deployment. As an example, consider a WSN that is deployed by dropping sensor nodes from an aircraft. Since their position, neighbourhood and the environmental conditions are unknown before deployment, the utility functions that model the information gain received from their interactions need to be learnt online.

Crucially, in stochastic and *a priori* unknown environments, agents are no longer faced with a one shot optimisation problem as encoded in a canonical DCOP. Rather, we now have a problem in which agents have to coordinate to solve a sequence of "DCOP-like" problems in order to simultaneously reduce uncertainty about the local utility functions (*exploration*) and maximise the global utility (*exploitation*). This implies the need for striking a balance between exploration and exploitation. Focusing solely on exploration results in certainty about the agents' environment, but wastes resources by taking suboptimal actions. Similarly, consistently taking the joint action that is currently believed to be the best is also suboptimal because this belief might be incorrect.

To date, this challenge has not been satisfactorily addressed by the DCOP community; existing DCOP algorithms either do not consider the trade off between exploration and exploitation, or fail to make this trade off in a principled and efficient manner. Indeed, most existing DCOP algorithms, such as ADOPT, DPOP, and max-sum, are not able to represent either stochastic or unknown functions [13, 5, 12, 15, 14]. In particular, other local approximate algorithms have been proposed for problems with functions that are *a priori* unknown but non-stochastic [17, 6], or stochastic but with *a priori* knowledge about the un-

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

derlying distributions [2]. More recently, the E-DPOP algorithm has been proposed for solving DCOPs with utility functions whose values are influenced by exogenous sources of stochasticity. These sources are modelled by random variables, whose underlying probability distributions are either known [10] or unknown [9]. Whilst a significant contribution to the field, it divides exploration and exploitation into two distinct phases, a process that is known to make its performance dependent on the specific problem instance [16]. Moreover, as acknowledged by the authors, E-DPOP is an incomplete algorithm when applied to non-linear evaluation functions, and as a result can perform arbitrarily poorly on general problem instances [10].

In contrast, the multi-armed bandit (MAB) community has addressed the trade off between exploration and exploitation in a principled fashion from a single agent perspective [8, 3, 18]. A MAB is a simple analytical tool for modelling decision making under uncertainty. In more detail, a MAB is a slot machine with multiple arms, each of which yields a reward, drawn from an unknown but fixed probability distribution. The aim of the problem is to sequentially pull the arms so as to maximise the cumulative reward over a finite time horizon. To achieve this, a number of computationally efficient pulling algorithms have been proposed, such as ε-first [4], ε-greedy [16] and upper confidence bound (UCB) [3]. Whereas the former two are sensitive to the choice of the ε parameter, the latter provides optimal theoretical guarantees on the regret (the difference between its performance and that of the theoretical optimal solution) without any need for parameter tuning.

Thus, to address the shortcoming of canonical DCOPs, we develop HEIST[1], which combines the robustness and scalability of decentralised coordination with the optimal exploration/exploitation trade off that the MAB algorithms provide. HEIST solves MAB–DCOPs, an extension of canonical DCOPs, in which each local utility function becomes a MAB. This effectively models both the stochastic nature of realistic decentralised coordination problems, as well as the absence of *a priori* knowledge. Unlike a DCOP, a MAB–DCOP is not a single shot optimisation problem, but rather a sequential problem in which agents need to coordinate their joint actions over multiple time steps, so as to maximise the cumulative global utility received over a finite time horizon. HEIST achieves this by repeatedly choosing the joint action with the highest estimated upper confidence bound (UCB) on the sum of local utilities received in a single time step, which is a non-linear combination of the confidence bounds on the local utilities. HEIST optimally computes this joint action using a message passing algorithm known as generalised distributive law (GDL) [1]. The GDL algorithm has been shown to be very efficient for solving various factorisable optimisation problems, such as the one we face in this paper. By using the GDL to maximise the UCB in a decentralised fashion, HEIST is computationally efficient, and provides optimal asymptotic bounds on the regret of the global cumulative performance.

In more detail, this paper contributes to the state of the art as follows:

- We introduce MAB–DCOPs, a new formalism to represent decentralised coordination problems with stochasticity and the absence of *a priori* knowledge about local utility functions.

[1]HEIST coordinates a group of bandits, hence its name.

- We develop HEIST, a novel algorithm to solve MAB–DCOPs, and prove that it provides optimal asymptotic bounds on the regret of the global cumulative utility.
- We empirically evaluate HEIST in a reproducible controlled environment, and show that it outperforms other state of the art techniques from the MAB and DCOP literature (among which max-sum and ε-first) by up to 1.5 orders of magnitude on MAB–DCOPs.

The remainder of this paper is structured as follows. In Section 2 we discuss related work on MABs and DCOPs. In Section 3 we formally define MAB–DCOPs. In Section 4, we present HEIST, empirically evaluate it in Section 5, and conclude in Section 6.

2. PRELIMINARIES

As discussed in the introduction, our approach lies on the nexus of DCOPs and MABs. Therefore, in this section we discuss these two bodies of literature in more detail.

2.1 DCOPs

Decentralised coordination problems can be encoded as DCOPs, which are defined as follows:

DEFINITION 1. *A DCOP is a tuple $\langle A, X, D, U, F\rangle$ where:*

- *$A = \{1, \dots, |A|\}$ is a set of agents.*
- *$X = \{x_1, x_2, \dots, x_n\}$ is a set of variables.*
- *$D = \{D_1, \dots, D_n\}$ is a set of finite domains, where D_i is the domain of variable x_i.*
- *$F : X \to A$ is a function that assigns variables to agents. Each agent controls (the value of) the variables that are assigned to it.*
- *$U = \{U_1, \dots, U_m\}$ is a set of local utility functions defined over a local scope $\mathbf{x}_j \subseteq X$, which assigns a real value to each assignment to $\mathbf{x}_j$.*[2]

The solution of a DCOP is an assignment X^* to variables X that maximises the sum of the utility functions:

$$X^* = \arg\max_X \sum_{j=1}^{m} U_j(\mathbf{x}_j) \tag{1}$$

Many algorithms have been developed to solve DCOPs. Some of these [15, 14] exploit the generalised distributive law (GDL) to achieve computation and communication efficiency [1]. The GDL message passing algorithm exploits the factorisability of a broad class of optimisation problems (which includes DCOPs) in order to solve them in an efficient and decentralised manner. A defining property of these problems is that the valuation algebra of their global objective function is a *commutative semi-ring*, an algebraic structure which is defined as follows:

DEFINITION 2. *A commutative semi-ring is a triple $\langle R, \oplus, \otimes\rangle$, where R is a non-empty set and $\oplus$ and $\otimes$ are two (abstract) binary associative and commutative operators over R, such that $\otimes$ distributes over $\oplus$. Furthermore, there exists an identity element $\mathbf{0} \in R$ such that $x \oplus \mathbf{0} = x$ for all $x \in R$, and an identity element $\mathbf{1} \in R$ such that $x \otimes \mathbf{1} = x$ for all $x \in R$.*

[2]In the DCOP literature, these are also known as constraint functions.

The objective functions of the typology of problems solved by the GDL can be defined in terms of operators $\oplus$ and $\otimes$, a set of variables X and a set of functions U, similar to those in Definition 1. They can be encoded as *factor graphs* [7], undirected bipartite graphs in which vertices represent variables X and functions U, and edges encode the "is a parameter of" relation. The decentralised GDL message passing algorithm operates directly on a factor graph, and consists of two separate algorithms (Algorithms 1 and 2), one for each type of factor graph vertex. Algorithm 1 performs the computation associated with variables, and is executed by agent $F(x_i)$ that controls variable x_i, while Algorithm 2 performs the computation associated with functions, and is executed by one of the agents whose variables are a parameter of U_j.

Algorithm 1 The GDL algorithm for variable x_i. $\mathcal{M}(i)$ is the set of indices of neighbouring functions. $R_{j\to i}(x_i)$ is a message from function U_j computed in Algorithm 2

```
1: procedure GDL_VARIABLE(i)
2:     while stopping condition has not been met do
3:         for all $j \in \mathcal{M}(i)$ do        ▷ For all adjacent functions
4:             if $|\mathcal{M}| = 1$ or no messages received yet then
5:                 Send $Q_{i\to j}(x_i) = \mathbf{1}$ to $U_j$
6:             else
7:                 Send $Q_{i\to j}(x_i) = \bigotimes_{k\in\mathcal{M}(i)\setminus j} R_{k\to i}(x_i)$ to $U_j$
8:             end if
9:         end for
10:        Wait for new messages from all $U_j : j \in \mathcal{M}(i)$
11:    end while
12:    return $Z_i(x_i) = \bigotimes_{j\in\mathcal{M}(i)} R_{j\to i}(x_i)$
13: end procedure
```

Algorithm 2 The GDL algorithm for function U_j. $\mathcal{N}(j)$ is the set of indices of neighbouring variables. $Q_{i\to j}(x_i)$ is a message from variable x_i computed in Algorithm 1.

```
1: procedure GDL_FUNCTION(j)
2:     while stopping condition has not been met do
3:         Wait for new messages from all $x_i : i \in \mathcal{N}(j)$
4:         for all $i \in \mathcal{N}(j)$ do        ▷ For all adjacent variables
5:             Send to $x_i$ message $R_{j\to i}(x_i) =$
                   $\bigoplus_{\mathbf{x}_j\setminus x_i} \left( U_j(\mathbf{x}_j) \otimes \bigotimes_{k\in\mathcal{N}(j)\setminus i} Q_{k\to j}(x_k) \right)$
6:         end for
7:     end while
8: end procedure
```

The following theorem is a fundamental property of the GDL message passing algorithm:

THEOREM 1. *If the factor graph is acyclic and the stopping criterion is chosen such that the algorithm is run for a number of iterations equal to the diameter of the factor graph, the following equation holds for each variable $x_i \in X$:*

$$Z_i(x_i) = \bigoplus_{X\setminus x_i} \bigotimes_{j=1}^{m} U_j(\mathbf{x}_j) \tag{2}$$

For proof of this theorem, see [11] (Chapter 26) and [1] (Theorem 3.1). The same result can be obtained for cyclic factor graphs by first transforming these into junction trees and using a slightly modified formulation of this algorithm [1]. For ease of exposition, in this paper we only consider acyclic factor graphs, in the knowledge that our algorithms and results also apply to junction trees with minimal modifications.

By instantiating the GDL algorithm for the max-sum commutative semi-ring $\langle \mathbb{R}, \max, + \rangle$, we obtain an algorithm for solving DCOPs (this algorithm is known as max-sum [15]). To see why this is the case, note that Equation 2 becomes:

$$Z_i(x_i) = \max_{X\setminus x_i} \sum_{j=1}^{m} U_j(\mathbf{x}_j) \tag{3}$$

This is the *maximum marginal* global utility that can be obtained for each assignment to variable x_i. As a direct consequence of this, setting $x_i^* = \arg\max_{x_i} Z_i(x_i)$ yields the variable assignment that maximises Equation 1. Note that this is only the case if the optimal solution is unique. If not, the solution can be made unique by adding small random values to the utility functions [15] (this is the method used in our experiments), or an additional utility propagation phase may be used [14]. We return to the GDL algorithm in Section 4 when we present HEIST, which uses the GDL algorithm instantiated for a special semi-ring designed to maximise the upper confidence bound on received utility.

2.2 Multi–Armed Bandits

A MAB is a slot machine with K arms, each of which delivers rewards drawn from an unknown distribution. The agent's goal is to choose which arms to pull so as to maximise expected cumulative reward over a finite time horizon T. In more detail, let P be a pulling policy (a sequence of pulls), and $i(t)$ denote arm chosen at time t by P, and $r_{i(t)}(t)$ the reward received by pulling that arm at time t. Then, we can formalise the agent's goal as follows:

$$P^* = \arg\max_{\mathbf{P}} \sum_{t=1}^{T} r_{i(t)}(t) \tag{4}$$

where $\mathbf{P}$ is the set of all possible policies. Clearly, if the reward distributions of each arm were known, the optimal policy would be to always pull the arm with the highest expected reward. This hypothetical scenario sets a performance benchmark known as *regret* against which to compare any policy. The regret $R_P(T)$ of a policy P after T time steps is the difference between the expected reward of the theoretical optimal policy and that obtained by P:

$$R_P(T) = \sum_{t=1}^{T} [\mu^* - \mu(i(t))] \tag{5}$$

where $\mu(i)$ is the *expected reward* of arm i and $\mu^* = \max_i \mu(i)$.

As mentioned in the introduction, there exist a number of pulling policies for minimising regret. Among these, UCB is one of the most widely used, since it is non-parametric and achieves asymptotically optimal regret. In more detail, UCB pulls each arm once at the beginning, then at each subsequent time step t, UCB selects arm $i^*(t)$ with the maximum upper confidence bound on the expected reward:

$$i^*(t) = \arg\max_{i\in[1,K]} \left[\hat{\mu}(i,t) + \sqrt{2 \ln t \frac{(u_{\max} - u_{\min})^2}{n(i,t)}} \right] \tag{6}$$

where $\hat{\mu}(i,t)$ is the sample mean of the rewards of arm i received until t, and $n(i,t)$ is the number of times UCB pulled arm i before time step t. UCB assumes the support of the reward function is bounded, i.e. $r_i(t) \in [u_{\min}, u_{\max}]$.

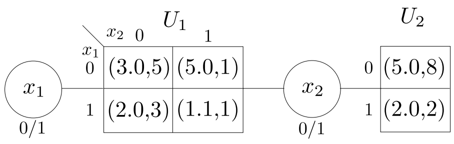


Figure 1: The MAB–DCOP from Example 1

The two terms in Equation 6 determine the trade off between exploration and exploitation. The larger the first, the more *exploitation* is favoured, since it is an estimate of the expected reward of arm i. The larger the second, the more *exploration* is favoured, since it represents the uncertainty in this estimate. In Section 4, we show how HEIST generalises the UCB algorithm to maximise the sum of rewards received from multiple (local) MABs in a MAB–DCOP and trades off exploration and exploitation in a decentralised fashion.

3. MAB–DCOPS

A MAB–DCOP is a DCOP where each utility function U_j is replaced by a MAB, such that each joint assignment $\mathbf{x}_j \in D_{\mathbf{x}_j}$ of the agents connected to that MAB becomes an arm of that bandit. Thus, in a MAB–DCOP, there is no *a priori* knowledge about the utility functions that govern the agents' interactions, and these interactions are subject to stochasticity. In more detail, for each $j \in \{1, \dots, m\}$, the utility $U_j(\mathbf{x}_j)$ obtained by choosing $\mathbf{x}_j \in D_{\mathbf{x}_j}$ is drawn from an unknown, but fixed, distribution, with (unknown) expected value $\mu(\mathbf{x}_j)$, and bounded support $[u_{\min}, u_{\max}]$. The agents' goal is to choose an optimal joint assignment $X^*(t) = \langle x_1^*(t), \dots, x_n^*(t)\rangle$ at each time t, such that the expected cumulative utility over a finite time horizon T is maximised:

$$\begin{aligned}[X^*(1), \dots, X^*(T)] &= \operatorname*{arg\,max}_{X(1),\dots,X(T)} \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{m} U_j(\mathbf{x}_j(t))\right] \\ &= \operatorname*{arg\,max}_{X(1),\dots,X(T)} \sum_{t=1}^{T}\sum_{j=1}^{m} \mu(\mathbf{x}_j(t)) \qquad (7)\end{aligned}$$

where $\mathbf{x}_j(t)$ is the chosen assignment to $\mathbf{x}_j$ at time t. To illustrate MAB–DCOPs, consider the following example.

EXAMPLE 1. *Consider the MAB–DCOP in Figure 1 with two binary variables x_1 and x_2, and two functions $U_1(\mathbf{x}_1)$ and $U_2(\mathbf{x}_2)$, at time $t = 10$. Utility functions are represented as tables, each cell of which has a tuple $(\hat{\mu}(\mathbf{x}_j, t), n(\mathbf{x}_j, t))$, where $\hat{\mu}(\mathbf{x}_j, t)$ is the sample mean of the utility received for the assignment $\mathbf{x}_j$ at t, and $n(\mathbf{x}_j, t)$ is the number of times that assignment has been sampled. Given the current sample means, assignment $(x_1 = 0, x_2 = 0)$ seems to be optimal. However, $(x_1 = 0, x_2 = 1)$ could be the real optimal, since $U_1(0, 1)$ and $U_2(1)$ have only been sampled at most twice.*

As this example suggests, to solve MAB–DCOPs, agents need to coordinate over the assignments to variables X at each time step, in order to trade off exploration (reducing uncertainty about the expected utility of each joint assignment) and exploitation (using the joint assignment that is believed to maximise reward).

Similar to MABs, we can define the regret of a coordination algorithm in a MAB–DCOP as a measure of the performance of a particular algorithm. In more detail, let $A = X^A(1), X^A(2), \dots$ denote a coordination algorithm that chooses joint assignment $X^A(t) = \langle x_1^A(t), \dots, x_n^A(t)\rangle$ at time t. The regret $R_A(T)$ for T time steps can be formalised as:

$$R_A(T) = T \cdot \max_X \sum_{j=1}^{m} \mu(\mathbf{x}_j) - \sum_{t=1}^{T}\sum_{j=1}^{m} \mu\left(\mathbf{x}_j^A(t)\right) \qquad (8)$$

HEIST, which is described next, is an algorithm that enables agents to make the trade off between exploration and exploitation in a principled manner by provably achieving an asymptotically optimal regret.

4. THE HEIST ALGORITHM

In this section, we present HEIST, an algorithm for solving MAB–DCOPs with asymptotically optimal regret. Using HEIST, agents coordinate at each time step to identify the best joint assignment to variables X, i.e. the arms that should be pulled on the local MABs (functions U) to minimise regret over time horizon T. In more detail, at each time step t, HEIST uses a GDL message passing phase to find the joint assignment $X^*(t)$ that maximises the UCB on the received utility. The formulation of this UCB is different from the UCB given in Equation 6—in fact, it is a generalisation—since the objective in a MAB–DCOP is to maximise the sum of rewards of *multiple* local MABs, instead of a single MAB:

$$X^*(t) = \operatorname*{arg\,max}_X \left[\sum_{j=1}^{m} \hat{\mu}(\mathbf{x}_j, t) + \sqrt{2 \ln t \sum_{j=1}^{m} \frac{(u_{\text{range}})^2}{n(\mathbf{x}_j, t)}}\right] \qquad (9)$$

Here, $\hat{\mu}(\mathbf{x}_j, t)$ is the sample mean at time t of the utility obtained by assignment $\mathbf{x}_j$ from MAB U_j, $n(\mathbf{x}_j, t)$ is the number of times a specific assignment to $\mathbf{x}_j$ was made, and $u_{\text{range}} = u_{\max} - u_{\min}$.

EXAMPLE 2. *In Example 1, $X^*(10) = (x_1 = 0, x_2 = 1)$, with a UCB equal to $7 + \sqrt{2\ln(10)\left(1 + \frac{1}{2}\right)}$, assuming $u_{\text{range}} = 1$.*

Calculating joint action $X^*(t)$ in Equation 9 is not trivial. For instance, this problem cannot be solved using a DCOP algorithm, since the objective function is not decomposable into a sum of factors (i.e. it is non-linear). As a result, the application of a canonical DCOP algorithm to this problem can lead to sub-optimality [10] (which we will show in the empirical evaluation). In contrast, HEIST is optimal by applying GDL to the GDL–UCB semi-ring, a special semi-ring, which is guaranteed to preserve the joint assignment with the optimal UCB.

Now, at each time t, HEIST proceeds in two steps. First, it uses GDL instantiated for the GDL–UCB semi-ring to compute the *maximum marginal* UCB of each variable assignment — the maximum UCB that can be achieved for each assignment to an individual variable—in a decentralised fashion. Second, each agent uses the result of the first step to choose the variable assignments for the variables it controls that maximise the global UCB in Equation 9.

Before proceeding, with slight abuse of notation, we change the signature of local utility functions to output tuples of the form: $U_j(\mathbf{x}_j, t) = (\hat{\mu}(\mathbf{x}_j, t), b(\mathbf{x}_j, t)^2)$. Here, $\hat{\mu}(\mathbf{x}_j, t)$ is the sample mean of assignment $\mathbf{x}_j \in D_{\mathbf{x}_j}$ at time t and $b(\mathbf{x}_j, t) = u_{\text{range}}\sqrt{2\ln t / n(\mathbf{x}_j, t)}$ is its (local) upper confidence bound t (cf. the two terms in Equation 6).

HEIST is split up into two algorithms, one for variables (Algorithm 3) and one for functions (Algorithm 4). Before

Algorithm 3 The HEIST message passing algorithm for variable x_i

```
1: $t_{\min} := \max_{k\in[1,m]} |D_{\mathbf{x}_k}|$   ▷ Wait for initial samples (line 7)
2: for $t = t_{\min} + 1$ to $T$ do   ▷ For each joint pull
3:     $Z_i(x_i) = $ GDL_VARIABLE($i$)   ▷ See Algorithm 1
4:     Set $x_i^*(t) := \arg\max_{x_i} \left[ \max_{(\hat{\mu},b^2)\in Z_i(x_i)} (\hat{\mu} + b) \right]$
5:     Send message sample($x_i^*(t)$) to all $U_j : j \in \mathcal{M}(i)$
6: end for
```

Algorithm 4 The HEIST message passing algorithm for function U_j. Line numbering continued from Algorithm 3.

```
7: for each $\mathbf{x}_j \in D_{\mathbf{x}_j}$, sample $U_j(\mathbf{x}_j)$ once
8: $t_{\min} := \max_{k\in[1,m]} |D_{\mathbf{x}_k}|$
9: for $t = t_{\min} + 1$ to $T$ do   ▷ For each joint pull
10:     execute GDL_FUNCTION($j$)   ▷ See Algorithm 2
11:     Wait for sample($x_i^*(t)$) messages from all $x_i : i \in \mathcal{N}(j)$
12:     Pull arm $\mathbf{x}_j^*(t) = \{x_i^*(t) \mid i \in \mathcal{N}(j)\}$
13:     Update sample mean $\hat{\mu}(\mathbf{x}_j^*, t)$ and sample count $n(\mathbf{x}_j^*, t)$
14: end for
```

communication commences, functions first sample the utility for their local domains (line 7) which takes $\max_{k\in[1,m]} |D_{\mathbf{x}_k}|$ time steps. This is analogous to the UCB algorithm, which pulls each arm once at the start. Then, functions and variables execute the GDL message passing algorithm (lines 2 and 8) instantiated for the GDL–UCB semi-ring, which is defined as follows:[3]

DEFINITION 3. *The GDL–UCB semi-ring is a semi-ring $\langle R, \max_{\succ}, +\rangle$ such that:*

- $R = \mathcal{P}(\mathbb{R} \times \mathbb{R})$ *a set of* sets[4] *of tuples, which have the same signature as the tuples output by functions U_j—the first element of each tuple is a sample mean $\hat{\mu}$, and the second is a squared bound b^2. The identity elements are $\mathbf{0} = \{(-\infty, -\infty)\}$ and $\mathbf{1} = \{(0, 0)\}$.*
- $\max_{\succ}$ *is an operator that takes multiple sets $S_1, \ldots, S_k \subseteq R$ as input and outputs a set S' such that:*
$$S' = \max_{\succ}(S) = \left\{ s \in \bigcup_{i=1}^{k} S_i \mid \forall s' \in \bigcup_{i=1}^{k} S_i : s' \not\succ s \right\}$$
Operator $\max_{\succ}$ *filters out so-called* dominated *tuples from sets $S_1, \ldots, S_k$—those that cannot maximise the global UCB. Domination is formally defined by binary operator $\succ$ such that $(\hat{\mu}_1, b_1^2) \succ (\hat{\mu}_2, b_2^2)$ iff:*
$$(\hat{\mu}_1 - \hat{\mu}_2 > b_2 - b_1) \wedge$$
$$\left(\hat{\mu}_1 - \hat{\mu}_2 > \sqrt{b_2^2 + u_{\text{range}}^2 2\ln t} - \sqrt{b_1^2 + (u_{\text{range}})^2 2\ln t}\right) \wedge$$
$$\left(\hat{\mu}_1 - \hat{\mu}_2 > \sqrt{b_2^2 + u_{\text{range}}^2 \frac{2\ln t}{t}} - \sqrt{b_1^2 + (u_{\text{range}})^2 \frac{2\ln t}{t}}\right)$$
- $+$ *is a binary operator such that if $S_1, S_2 \in R$, then $S_1 + S_2 = \{(\hat{\mu}_1 + \hat{\mu}_2, b_1^2 + b_2^2) \mid \forall(\hat{\mu}_1, b_1^2) \in S_1, \forall(\hat{\mu}_2, b_2^2) \in S_2\}$. Thus, $+$ sums all pairs of tuples in S_1 and S_2.*

As a consequence of the non-linearity of the objective function in Equation 9, choosing the assignment that maximises the sum of UCBs on local utility, does not (necessarily) produce optimality of the UCB on the sum of local utilities. Thus, we cannot simply discard one tuple $(\hat{\mu}_1, b_1^2)$ in favour of tuple $(\hat{\mu}_2, b_2^2)$ if $\hat{\mu}_1 + b_1 < \hat{\mu}_2 + b_2$. This problem is addressed by the definition of the $\max_{\succ}$ operator, which is designed to discard only those tuples that are *guaranteed* to lead to global sub-optimality. As a result, it imposes a partial order over tuples to preserve the tuple that produces global optimality.

Executing the GDL algorithm on the GDL–UCB semi-ring yields the marginal function $Z_i(x_i)$ for each variable x_i (line 3). It can be proved that for each assignment $x_i \in D_i$, the set $Z_i(x_i)$ contains a tuple $(\hat{\mu}, b^2)$, such that $\hat{\mu} + b$ is the maximum achievable global UCB given that assignment (Theorem 2). Using this marginal function $Z_i(x_i)$, the second phase of HEIST first computes the maximum UCB for each assignment x_i (line 4, expression between brackets) and then selects the assignment with the maximum UCB (remainder of line 4). Then, in line 5, each variable informs adjacent functions of its assignment, after which all functions sample assignments (line 12).

EXAMPLE 3. *The following demonstrates the operation of* HEIST *on a single time step ($t = 10$) of the MAB–DCOP from Example 1. Let $c = 2\ln(10)$, and $u_{\text{range}} = 1$.*

GDL Iteration 1*:*

$$Q_{1\to1}(x_1) = Q_{2\to1}(x_2) = Q_{2\to2}(x_2) = \{(0, 0)\}$$
$$\begin{array}{ll} R_{1\to1}(0) = \{(5, c)\}, & R_{1\to1}(1) = \{(2, c/3), (1.1, c)\} \\ R_{1\to2}(0) = \{(3, c/5)\}, & R_{1\to2}(1) = \{(5, c)\} \\ R_{2\to2}(0) = \{(5, c/8)\}, & R_{1\to2}(1) = \{(2, c/2)\} \end{array}$$

GDL Iteration 2*:*

$$Q_{1\to1}(x_1) = (0, 0), Q_{2\to1}(x_1) = R_{2\to2}(x_1), Q_{2\to2}(x_1) = R_{2\to1}(x_1)$$
$$\begin{array}{ll} R_{1\to1}(0) = \{(7, c(1 + 1/2))\}, & R_{1\to1}(1) = \{(7, c(1/3 + 1/8))\} \\ R_{1\to2}(0) = \{(3, c/5)\}, & R_{1\to2}(1) = \{(5, c)\} \\ R_{2\to2}(0) = \{(5, c/8)\}, & R_{1\to2}(1) = \{(2, c/2)\} \end{array}$$

At this point, GDL has converged. We can now calculate the marginals $Z_i(x_i)$:

$$\begin{array}{ll} Z_1(0) = (7, 2\ln(10)(1 + 1/2)), & Z_1(1) = (7, 2\ln(10)(1/3 + 1/8)) \\ Z_2(0) = (8, 2\ln(10)(1/5 + 1/8)), & Z_2(1) = (7, 2\ln(10)(1 + 1/2)) \end{array}$$

By calculating the UCB associated with these tuples, we obtain $x_1^* = 0, x_2^* = 1$, with a UCB of $7 + \sqrt{2\ln(10)\left(1 + \frac{1}{2}\right)}$, which indeed maximises Equation 9 (cf. Example 2).

For the GDL message passing phase of HEIST, we can derive the following result:

THEOREM 2 (MAIN RESULT 1). *If the factor graph is acyclic and the stopping criterion is chosen such that GDL message passing phase is run for a number of iterations that is equal to the diameter of the factor graph, the following equation holds for each $t > \max_{k\in[1,m]} |D_{\mathbf{x}_k}|$:*

$$\max_{(\hat{\mu},b^2)\in Z_i(x_i)} (\hat{\mu}+b) = \max_{X\setminus x_i} \left[\sum_{j=1}^{m} \hat{\mu}(\mathbf{x}_j, t) + \sqrt{2\ln t \sum_{j=1}^{m} \frac{(u_{\text{range}})^2}{n(\mathbf{x}_j, t)}} \right]$$

Put differently, Theorem 2 states that, after the initial pulls, set $Z_i(x_i)$ contains the tuple that yields the *marginal maximum UCB* that can be obtained for each assignment to x_i, $i \in [1, n]$. As a direct consequence, line 12 pulls the arm on each function with the highest overall UCB (Equation 9). This observation leads to the following theorem:

[3]Proofs of correctness and explanation of the operators can be found in the Appendix.
[4]Here, $\mathcal{P}(S)$ denotes the powerset of set S.

THEOREM 3 (MAIN RESULT 2). *Suppose the factor graph is acyclic and the stopping criterion is chosen such that GDL message passing phase is run for a number of iterations that is equal to the diameter of the factor graph at every time t. Let $X^{\mathbb{E}} = \arg\max_X \sum_{j=1}^{m} \mu(\mathbf{x}_j)$ be the joint assignment that maximises the (unknown) expected utility. Let $d(X) = \mu(X^{\mathbb{E}}) - \mu(X)$ denote the difference between the expected utility of the optimal action $X^{\mathbb{E}}$ and that of a particular joint action X. Given this, the cumulative regret $R_{\text{HEIST}}(T)$ of HEIST after T time steps is at most:*

$$\sum_{X \neq X^{\mathbb{E}}} \frac{u_{\text{range}} 8 \ln T}{d(X)} + \left(1 + \frac{\pi^2}{3}\right) \sum_{X \neq X^{\mathbb{E}}} d(X) \qquad (10)$$

Based on this result, we can show that HEIST provides asymptotically optimal regret bounds, by comparing against best achievable regret:

THEOREM 4 (MAIN RESULT 3). *For any algorithm, there exists a constant $C \geq 0$, and a particular instance of the MAB–DCOP problem, such that the regret of that algorithm within that particular problem is at least $C \ln T$.*

Thus, the regret bound of HEIST (Equation 10) only differs from the best possible with a constant factor. The proofs of the theorems can be found in the Appendix.

5. EMPIRICAL EVALUATION

In the previous section, we proved that the regret achieved by HEIST is guaranteed to be a constant factor away from the optimal. However, further empirical analysis is needed to gauge HEIST's practical performance, in terms of solution quality as well as communication and computation overhead. Moreover, such analysis can focus on the algorithm's performance when the GDL phase of the algorithm is not run until convergence, one of the conditions for optimality in Theorem 3. Instead, the number of iterations c of the GDL phase can be a parameter for tuning the trade off between solution quality and overhead (i.e. computation and communication). Note that it is not the objective of these experiments to study the properties of MAB–DCOPs and HEIST across all possible probability distributions, and instantiations of the utility function U. Due to space constraints, we would not be able to do justice to the requirements of different application domains, and the specific configurations of HEIST. This is left for future work.

Therefore, in this section, we benchmark several versions of HEIST against existing approaches, taken from the state of the art in the MAB and DCOP literature. Specifically, we compare HEIST against the following algorithms:

HEIST-c a version of HEIST where the GDL message passing phase is run for c iterations. When taking joint actions is cheap compared to communication, or when action is required before the GDL message passing phase is able to converge, c can be set to a value smaller than the diameter of the factor graph. In this case, optimality is no longer guaranteed, but it can lead to a good trade off between communication and solution quality.

ε-first an algorithm that samples from the utility functions (exploration) for the first εT time steps, and picks the one that is believed to be optimal (exploitation) for the remaining $(1-\varepsilon)T$ time steps. At the start of the exploitation phase, this algorithm runs a standard DCOP algorithm (max-sum) once to find the joint assignment that maximises the sum of the sample means of the local utilities. Using a DCOP algorithm in this way is equivalent to using E-DPOP [9] on a problem where each local assignment $\mathbf{x}_j$ for $j \in [1, m]$ is modelled by a single random variable. Thus, ε-first can be considered as E-DPOP applied to a MAB–DCOP. To perform well, ε needs to be tuned for each problem instance [3, 16]. After initial tests, we found that $\varepsilon = 0.02$ leads to good performance for the problems described below.

Monolithic UCB the (centralised) UCB algorithm that considers a MAB–DCOP as a single "monolithic" MAB with $\prod_{i=1}^{n} |D_i|$ arms.

Max-Sum a standard DCOP algorithm, applied to a DCOP in which the objective is to compute $X^+(t)$ at every $t \in [1, T]$, such that:

$$X^+(t) = \arg\max_X \sum_{j=1}^{m} \left[\hat{\mu}(\mathbf{x}_j, t) + \sqrt{\frac{(u_{\text{range}})^2 2 \ln t}{n(\mathbf{x}_j, t)}} \right] \qquad (11)$$

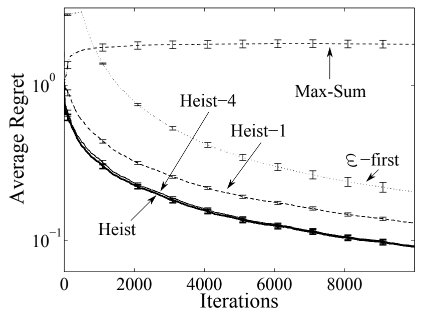
By solving this DCOP, the sum of UCBs on local utilities is maximised, instead of the UCB on the sum of utilities (Equation 9). Since the latter is clearly not linear, Equations 9 and 11 are not equivalent (except for $m = 1$). As a result, this decomposition leads to loss of optimality in the sense of Theorem 4. Léauté et al. observed a similar result for their (incomplete) algorithm when using a linear decomposition to maximise non-linear objective functions [10].

The ε-first and max-sum algorithms are included to demonstrate that standard DCOP algorithms are unsuitable for solving MAB–DCOPs, while the Monolithic UCB algorithm is included to demonstrate the need for exploiting the factorisability of a MAB–DCOP.

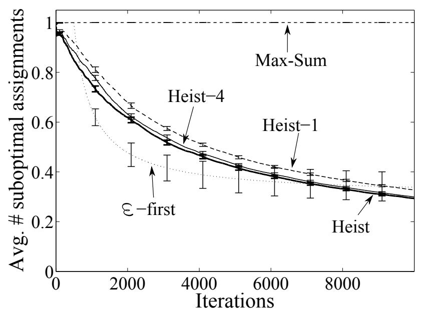
We randomly generated MAB–DCOP instances that can be encoded as acyclic factor graphs, with $n = 15$ and $m = 14$, and $|D_i| = 3$. Each utility function $U_j(\mathbf{x}_j)$ is governed by a set of normal distributions, one for each assignment to $\mathbf{x}_j$. In more detail, $U_j(\mathbf{x}_j) \sim \mathcal{N}(\mu(\mathbf{x}_j), \sigma^2(\mathbf{x}_j))$, where $\mu(\mathbf{x}_j)$ and $\sigma^2(\mathbf{x}_j)$ are uniformly drawn from intervals $[0, \mu_{\max}]$ and $[0, 1]$ respectively. Parameter $\mu_{\max}$ is used to control the relative importance that needs to be given to exploration and exploitation. When $\mu_{\max}$ is decreased, the received utilities become more noisy and the balance should be shifted towards exploration. Conversely, when $\mu_{\max}$ is increased, the optimal joint action becomes more easily identifiable, and agents start exploitation quite early on. For our experiments, we chose $\mu_{\max} = 1$ and $\mu_{\max} = 10$. These values were chosen during initial calibration to yield two sets of difficult problems, which simultaneously demonstrate the difference in required emphasis between exploration and exploitation.

The results are shown in Figures 2 and 3 for $\mu_{\max} = 1$ and $\mu_{\max} = 10$ respectively. Each algorithm was run 64 times on both problem classes to obtain statistically significant results. Error bars indicate the standard error of the mean. Monolithic UCB is omitted from all figures, because its regret converged approximately a factor of 3^{15} slower than HEIST. This was expectated, as it regards the problem as a MAB with 3^{15} arms, instead of 14 MABs with 9 arms each.

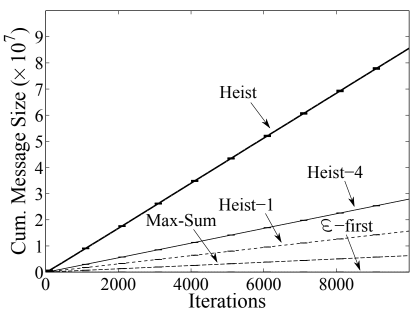
Now, Figures 2(a) and 3(a) show the average regret for the remaining algorithms, while Figures 2(b) and 3(b) show the number of average suboptimal assignments. As can be observed from these figures, HEIST and HEIST-4 outperform all others in terms of regret (up to 1.5 orders of magnitude for $\mu_{\max} = 10$). We found that for $c \geq 8$, the per-



(a) Regret

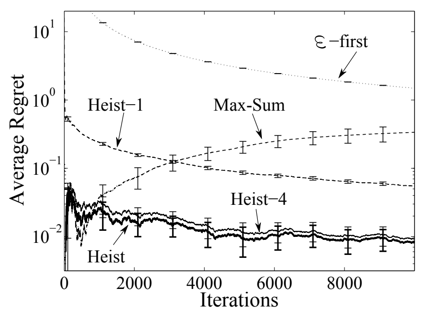


(b) Suboptimal joint assignments

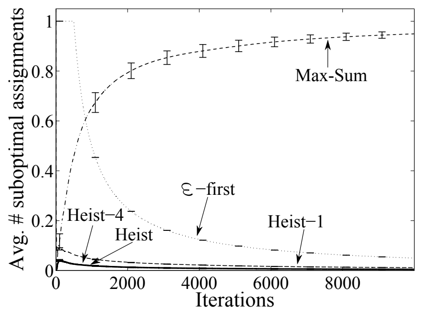


(c) Message size

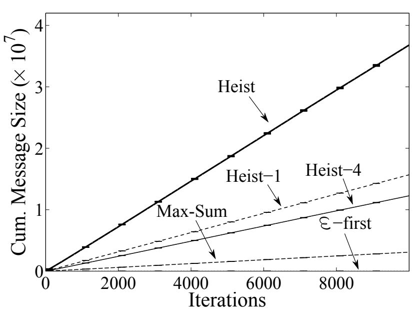
Figure 2: Empirical results for $\mu_{\max} = 1$



(a) Regret



(b) Suboptimal joint assignments



(c) Message size

Figure 3: Empirical results for $\mu_{\max} = 10$

formance of Heist (which was run for 30 iterations to ensure convergence) and Heist-c coincide, indicating that the algorithm performs well even when conditions of Theorem 2 are not met. Max-sum—and indeed any algorithm that solves Equation 11—is clearly suboptimal, as its regret does not converge to zero, and it consistently produces suboptimal assignments. The fact the regret of max-sum *increases* is counter intuitive. However, additional experimentation showed that for smaller problem instances, the difference between Heist and max-sum is much smaller, and their performance often coincides for problems with $m < 5$, while for $m = 50$, the difference in regret at $T = 10000$ was found to be more than 3 orders of magnitude. This leads us to believe that for larger problems, the non-linearity of Equation 9 is more pronounced, increasing the difference between the regret associated with assignments $X^+(t)$ and $X^*(t)$. The regret of the second DCOP based technique, ε-first, does converge to zero, but at a much slower pace than Heist. Based on these results, we can conclude that both DCOP techniques are unsuitable for solving MAB–DCOPs.

Focusing on communication overhead, Figures 2(c) and 3(c) show the cumulative message size expressed as the number of floating point values exchanged between the agents. Compared to ε-first (which needs a negligible number of messages to coordinate once) and max-sum (which exchanges scalars, instead of sets of tuples), Heist requires more communication. However, a good balance can be struck between solution quality and communication by reducing c to 4. Moreover, note that Heist requires each agent to exchange only 600 values per iteration of the MAB–DCOP, a value that is well within the capabilities of bandwidth constrained embedded agents. Finally, the computation required by Heist to solve problem instances with $n = 50$ and $T = 20000$ never exceeded 4 hours on a standard desktop PC, which is less than 200ms per agent per iteration. Again, this is well within the reach of embedded agents.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we developed Heist, a novel algorithm for coordination under stochasticity and the absence of *a priori* knowledge about the utility functions that govern the agents' interactions. Heist solves MAB–DCOPs, an extension to the canonical DCOP framework, in which the local utility functions are transformed into MABs. By so doing, a MAB–DCOP becomes a sequential problem, instead of a single-shot optimisation problem, in which agents' need to trade off exploration and exploitation in a decentralised fashion. We formalised this trade off as a problem of maximising the UCB on the global utility, which we showed is a non-linear objective function. While previous algorithms have been shown to be incomplete, i.e. are not guaranteed to maximise such functions, Heist is provably optimal. This is achieved by applying the GDL message passing algorithm on the GDL–UCB semi-ring, which is specifically designed to preserve the optimal joint variable assignment. We prove that the regret of Heist is asymptotically optimal, i.e. it only differs from the optimal achievable regret by a constant factor. In addition, empirical results demonstrate that Heist outperforms state of the art DCOP and MAB algorithms by up to 1.5 orders of magnitude.

For future work, we intend to further reduce the communication overhead of Heist. In the empirical results, we already applied a technique for achieving this (Heist-c), but this technique no longer carries the guarantee of optimality. Instead, we can let Heist automatically calibrate the amount communication to suit the level of dynamism in the environment, while maintaining optimality.

Acknowledgements This work was part of the ORCHID project (`http://www.orchid.ac.uk/`).

7. REFERENCES

[1] S. M. Aji and R. J. McEliece. The Generalized Distributive Law. *IEEE Trans. Inf. Theory*, 46(2):325–343, 2000.

[2] J. Atlas and K. Decker. Coordination for uncertain outcomes using distributed neighbor exchange. *AAMAS'10*, pages 1047–1054, 2010.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite–time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[4] E. Even-Dar, S. Mannor, and Y. Mansour. Pac bounds for multi–armed bandit and markov decision processes. *COLT'02*, pages 255–270, 2002.

[5] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In *Distributed Sensor Networks*, pages 257–295. Kluwer Academic Publishers, 2003.

[6] M. Jain, M. Taylor, M. Tambe, and M. Yokoo. DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. *IJCAI'09*, pages 181–186, 2009.

[7] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. on Information Theory*, 47(2):498–519, 2001.

[8] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Adv. in Appl. Math.*, 6(1):4–22, 1985.

[9] T. Léauté and B. Faltings. E[DPOP]: Distributed constraint optimization under stochastic uncertainty using collaborative sampling. *IJCAI–09 DCR Workshop*, pages 87–101, 2009.

[10] T. Léauté and B. Faltings. Distributed constraint optimization under stochastic uncertainty. *AAAI'11*, pages 68–73, 2011.

[11] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[12] R. J. Maheswaran, J. Pearce, and M. Tambe. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multiagent Systems*, pages 127–146. Springer-Verlag, 2005.

[13] P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180, 2005.

[14] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. *IJCAI'05*, pages 266 – 271, 2005.

[15] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artif. Intell.*, 175(2), 2011.

[16] R. S. Sutton and A. G. Barto, editors. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[17] M. Taylor, M. Jain, Y. Jin, M. Yokoo, and M. Tambe. When should there be a "Me" in "Team"?: Distributed multi-agent optimization under uncertainty. *AAMAS'10*, pages 109–116, 2010.

[18] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. *ECML'05*, pages 437–448, 2005.

Appendix – Proofs

PROOF SKETCH OF THEOREM 2. We first show that operator $\max_{\succ}$ filters out tuples that cannot maximise the global UCB. In particular, if there is at least one incoming message, then $(\hat{\mu}_2, b_2^2)$ is dominated by $(\hat{\mu}_1, b_1^2)$ if and only if for any incoming tuple $(\hat{\mu}_3, b_3^2)$, $\hat{\mu}_1 + \hat{\mu}_3 + \sqrt{b_1 + b_3} > \hat{\mu}_2 + \hat{\mu}_3 + \sqrt{b_2 + b_3}$ holds. This implies that $\hat{\mu}_1 - \hat{\mu}_2 > \sqrt{b_2 + b_3} - \sqrt{b_1 + b_3}$. By definition, $b_3 = u_{\text{range}}\sqrt{\frac{2\ln t}{n_3(t)}}$ at time t, where $n_3(t)$ is the number of samples included in $\hat{\mu}_3$. Note that $1 \le n_3(t) \le t$, and can take any arbitrary value within this interval. That is, $u_{\text{range}}^2 \frac{2\ln t}{t} \le b_3^2 \le u_{\text{range}}^2 2\ln t$. This implies that if at least one among the second or the third clause in the definition does not hold, then tuple $(\hat{\mu}_2, b_2^2)$ cannot be discarded (i.e. it is not dominated by tuple $(\hat{\mu}_1, b_1^2)$. In addition, if there are no incoming messages, then breaking the first clause indicates that $(\hat{\mu}_2, b_2^2)$ cannot be discarded either. That is, $(\hat{\mu}_2, b_2^2)$ is dominated by $(\hat{\mu}_1, b_1^2)$ iff all the clauses hold.

The proof of the claim that the GDL message passing phase yields $Z_i(x_i)$, the maximum marginal UCB for each assignment, follows a similar argument to that of Theorem 3.1 in [1], and thus, is omitted for brevity. □

PROOF SKETCH OF THEOREM 3. At each time t, after GDL message passing has converged, the joint assignment with the maximum UCB is chosen (see Theorem 2). Suppose that each time t, HEIST chooses joint assignment $X^*(t) = \langle x_1^*(t), \dots, x_n^*(t)\rangle$. In what follows, we estimate the expected number times $X^*(t) \neq X^{\mathbb{E}}$ is chosen, in order to estimate the regret of HEIST. In particular, let $N_T(X)$ denote the number of times HEIST chooses suboptimal joint assignment $X \neq X^{\mathbb{E}}$ before T. By estimating $N_T(X)$, we can estimate the number of times HEIST chooses a suboptimal joint assignment, and thus, derive a bound on its regret. That is,

$$R_{\text{HEIST}}(T) \le \sum_{X \in \prod D_i} N_T(X) d(X) \tag{12}$$

We provide an upper bound for $\mathbb{E}\left[N_T(X)\right]$ as follows. Note that $\mathbb{E}\left[N_T(X)\right]$ can be estimated by the following sum:

$$\mathbb{E}\left[N_T\left(X\right)\right] \le k + \sum_{t=1}^{T} P\left(X^*(t) = X, X \neq X^{\mathbb{E}}, N_{t-1}\left(X\right) \ge k\right) \tag{13}$$

The latter term can be further upper bounded by:

$$\sum_{t=1}^{T} P\left(\hat{\mu}(X(t),t) + b(X(t),t) \ge \hat{\mu}(X^{\mathbb{E}},t) + b(X^{\mathbb{E}},t), N_{t-1}\left(X\right) \ge k\right) \tag{14}$$

where $b(X,t) = \sqrt{2\ln(t)\sum_{j=1}^{m}\frac{(u_{\text{range}})^2}{n(\mathbf{x}_j,t)}}$. Intuitively, the probability that HEIST chooses $X^*(t) \neq X^{\mathbb{E}}$ can be bounded by the probability that $\hat{\mu}(X(t),t) + b(X(t),t) \ge \hat{\mu}(X^{\mathbb{E}},t) + b(X^{\mathbb{E}},t)$. This can be further bounded by:

$$\sum_{t=1}^{T}\sum_{s_j=k}^{t}\sum_{s=k}^{t} P\left(\hat{\mu}(X(s_j),s_j) + b\left(X\left(s_j\right),s_j\right) \ge \hat{\mu}(X^{\mathbb{E}},s) + b(X^{\mathbb{E}},s)\right) \tag{15}$$

Now, it is true that if $\hat{\mu}(X(s_j),s_j) + b\left(X\left(s_j\right),s_j\right) \ge \hat{\mu}(X^{\mathbb{E}},s) + b(X^{\mathbb{E}},s)$ holds then at least one of the following must hold:

1. $\hat{\mu}\left(X^{\mathbb{E}},s\right) + b\left(X^{\mathbb{E}},s\right) \le \mu\left(X^{\mathbb{E}},s\right)$.
2. $\mu\left(X(s_j),s_j\right) \le \hat{\mu}\left(X(s_j),s_j\right) + b\left(X(s_j),s_j\right)$.
3. $\mu\left(X^{\mathbb{E}},s\right) - \mu\left(X(s_j),s_j\right) \le 2b(X(s_j),s_j)$.

This can be shown by using similar argument to that of Theorem 1 in [3], and thus, is omitted for brevity. By using McDiarmid's inequality, we can show that both (1) and (2) hold with probability t^{-4}, and if $k \ge \lceil\frac{u_{\text{range}} 8\ln T}{d(X)}\rceil$, then (3) does not hold. This implies that:

$$\mathbb{E}\left[N_T\left(\mathbf{x}\right)\right] \le k + \sum_{t=1}^{T}\sum_{s=1}^{t}\sum_{s_j=1}^{t} 2t^{-4} \le k + \frac{\pi^2}{3} \tag{16}$$

for any $k \ge \lceil\frac{u_{\text{range}} 8\ln T}{d_{\mathbf{x}}}\rceil$. The last inequality is obtained from the Riemann Zeta Function for value of 2. Finally, substituting this into Equation 12 concludes the proof. □

PROOF SKETCH OF THEOREM 4. We can reduce all standard MAB problems to a MAB–DCOP with $m = 1$. According to [8], the best possible regret that an algorithm can achieve on standard MABs is $C\ln T$. Therefore, if there is an algorithm for MAB–DCOPs that provides better regret than $C\ln T$, then it also provides better regret bounds for standard MABs. □