AMSC 663 Mid-year Report: Task Assignment in a Human-Autonomous Image Labeling System

Addison Bohannon

December 14, 2015

Advisors:

Vernon J. Lawhern
Human Research and Engineering Directorate
US Army Research Laboratory
Aberdeen Proving Ground, MD 21005 USA
$v$ernon.j.lawhern.civ@mail.mil

Brian M. Sadler
Computational and Information Sciences Directorate
US Army Research Laboratory
Adelphi Laboratory Center, MD 20783 USA
$b$rian.m.sadler6.civ@mail.mil

Abstract

We want to design a human-autonomous system which can efficiently and accurately classify a database of images as interesting or not interesting. The system will iteratively distribute images for binary classification amongst multiple heterogeneous agents according to the Generalized Assignment Problem, and use the Spectral Meta-Learner, to combine multiple binary classification results in a principled manner. From the results, we hope to draw conclusions about the circumstances under which sensitive tasks can and should be delegated to autonomous technology in an optimal manner.

1 Introduction

Suppose that we wanted to sort a large database of images according to a simple binary classification such as interesting or not interesting. The defense and intelligence communities have this exact requirement. The sorting of satellite imagery requires trained professionals to scan images for indicators of valuable intelligence, triaging images for later thorough analysis. This is a costly and tedious task for humans. Certainly, there are means to reduce the workload for these intelligence analysts in the age of computer vision.

Sajda, et al. explore this very problem in [14]. They used computer vision algorithms to complement the efforts of a human performing image triage, sorting images as "interesting" or not. The group sought to synchronize the efforts of a single human performing Rapid Serial Visual Presentation (RSVP) with a single computer vision system. [1] Exploiting the singular ability of humans to quickly understand the "gist" of an image, the human agent quickly scanned the images. Images which elicited interesting responses through the RSVP paradigm were routed to the computer vision algorithm, which excels at object recognition. In essence, the system could intelligently sort the images according to the human analysis, and determine the source of the humans interest through the far quicker and more accurate object recognition skills of computer vision. Sajda, et al. additionally implemented this system in reverse, pruning the image database first through computer vision algorithms searching the images for predetermined interesting objects and routing the reduced database to the human for confirmation. Either method, autonomous technology as a mechanism for inference or as a filter, proved a viable complement to the single image analyst.

These results imply a more general approach to leveraging autonomous technology to support human efforts; however, when considering tasks with implications as serious as our national security, perhaps we should not entirely exclude humans from the decision loop. An obvious concern might be a lack of sensitivity of a computer vision algorithm. What if a valuable image were deemed not valuable and thus never seen by a human analyst? Just as likely is that a human is unusually inaccurate on a given day? Here, Ipierotis, et al. present a sensible recommendation: seek additional independent classifications from other agents when the classification outcomes of tasks have a non-negligible probability of error [6]. If one computer vision algorithm deems an image as interesting, confirm with another computer vision algorithm – or better yet, confirm with as many distinct agents as feasible. This implies that in order to reduce the workload for the human imagery analyst while increasing the overall accuracy of image labels, multiple autonomous and human agents must make independent classifications until a confident meta-classification is achieved.

Such a system is depicted in Figure 1; however, it begs two principal questions: how should images be distributed amongst the agents in such a system, and how should multiple labels for a single image be combined to make a decision about that image?

1.1 Problem

Let us simplify and formalize this system in order to formulate a clear problem statement. Figure 2, the Iterative Task Assignment System (ITAS), conceptualizes a system with n heterogeneous tasks and m heterogeneous agents. Let T be the set of tasks. Then, an agent performs the following binary decision: $Y : T \to \{-1, 1\}$. Denote $I = \{1, \ldots, n\}$ as the index of tasks and $J = \{1, \ldots, m\}$

[1] RSVP is a Brain-Computer Interface paradigm in which subjects passively view images at high rates of speed (2-10 Hz) while an electroencephalogram (EEG) measures surface potentials of brain activity. Using the oddball paradigm, novel stimuli, or interesting images, generate a neural signature in the EEG recordings which can be recognized through machine learning methods.

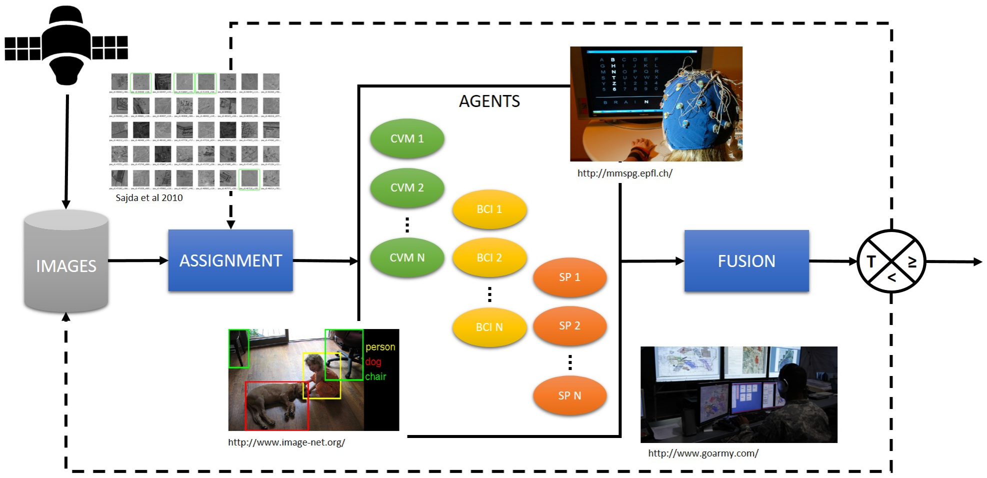

Figure 1: The image labeling system. An imaging sensor populates the database of images. An assignment node distributes images to agents in parallel. The agents perform binary classification – interesting or not interesting, and the results are consolidated at a fusion node. At this point, the confidence in the image classification label is used to threshold images for completion or routing back to the image database for re-assignment. Here, agents can be computer vision algorithms, RSVP subjects, or self-paced image analysts.

as the index of agents. Each task has an associated score parameter, $0 \leq s_i \leq 1$, which reflects the current confidence in the meta-label of that task, and each agent has a reliability, r_j, which reflects the agent's accuracy and a budget, b_j, which limits the workload for an agent on each iteration. The assignment of a task $i \in I$ to an agent $j \in J$ will have an associated value, $v_{ji} = f(s_i, r_j) \geq 0$ and cost, c_{ji}, which could reflect the time or resources required for task completion.

Accordingly, we want to determine:

1. Assignment Problem – How do we optimally assign images in each iteration in order to balance value and workload?
2. Joint Classification Problem – How do we infer the label of a task given a set of labels from multiple agents?

1.1.1 Task Assignment

This task assignment problem, where we are looking for the optimal assignment policy over all tasks and agents, $\{x_{ji}\}_{i\in I,\in J}$, can be mapped onto the Generalized Assignment Problem (GAP) as follows [8, 10]:

$$Z = \max_{\mathbf{x}} \sum_{i\in I} \sum_{j\in J} v_{ji} x_{ji} \tag{1}$$

1. $\sum_{i\in I} c_{ji} x_{ji} \leq b_j, \; j \in J$
2. $\sum_{j\in J} x_{ji} = 1, \; i \in I$
3. $x_{ji} \in \{0, 1\}$

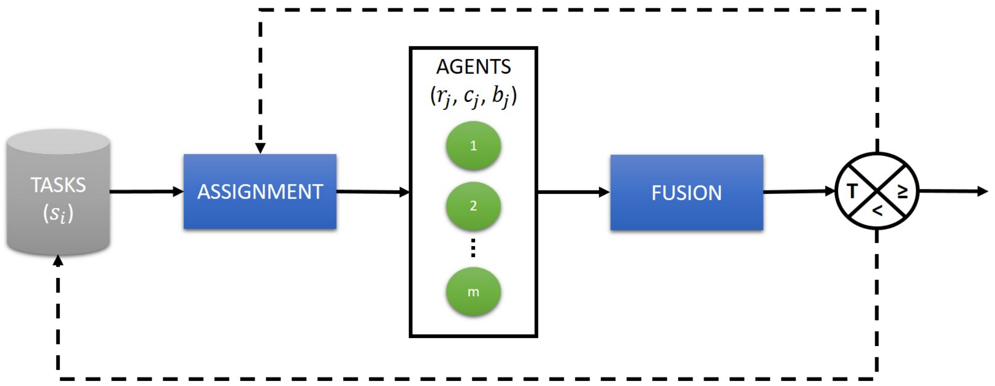

Figure 2: Iterative Task Assignment System. ITAS is a simplification of the image labeling system. Here, heterogeneous tasks, with an extrinsic confidence score, s_i, are assigned in parallel from a centralized node to heterogeneous agents, with the following intrinsic parameters: reliability, r_j, and budget, b_j. Additionally, each assignment has a cost, c_{ji}. The classification results are consolidated into a single joint classification at the fusion node, and the images are classified if the confidence in the meta-label exceeds a pre-set threshold, otherwise images are routed back to the database for re-assignment.

4. $c_{ji}, b_{ji} \in \mathbb{Z}_+$
5. $v_{ji} = g(r_j, s_j) \geq 0$

GAP is a NP-hard problem, but the decision problem is NP-complete [4, 10]. Solving GAP is at least as difficult as any problem in NP and has no known polynomial time solution algorithm, but verifying that a solution is feasible is computable in polynomial time [9]. This ability to verify solutions allows exact solution methods which seek to exhaustively search the solution space for a global maximum by confirming all possible solutions and keeping the global optimum. As the GAP is combinatorial, computational complexity of even the best exact methods can be prohibitive; therefore, commonly used exact methods do not evaluate the target function at all feasible solutions but rather use a heuristic to reduce the feasible set of solutions during the search [3, 8, 10].

1.1.2 Joint Classification

Given m agents which independently produce binary classifications on n tasks, we want to infer the label of each task without knowledge of the true labels of any of the tasks. The joint classification problem seeks to minimize the probability of error by finding a mapping, f, from a set of the independent classification labels for a task i from m agents, $\{\hat{y}_{ji}\}_{j \in J}$, to a meta-label, $\bar{y}_i$, such that the probability over all tasks of the meta-label being incorrect is minimized:

$$\arg\min_f \sum_{i \in I} \mathbb{P}(f(\hat{y}_{1i}, \ldots, \hat{y}_{mi}) \neq y_i) \tag{2}$$

1. $f : \{\hat{y}_{ji}\}_{j \in J} \to \bar{y}_i$
2. $\hat{y}_{ji},\ \bar{y}_i,\ y_i \in \{-1, 1\}$

The classification labels of all agents for all tasks can be captured in a matrix, $\mathbf{A} \in \{-1, 1\}^{m \times n}$, in which the rows correspond to agents and the columns correspond to tasks, see Figure 3.

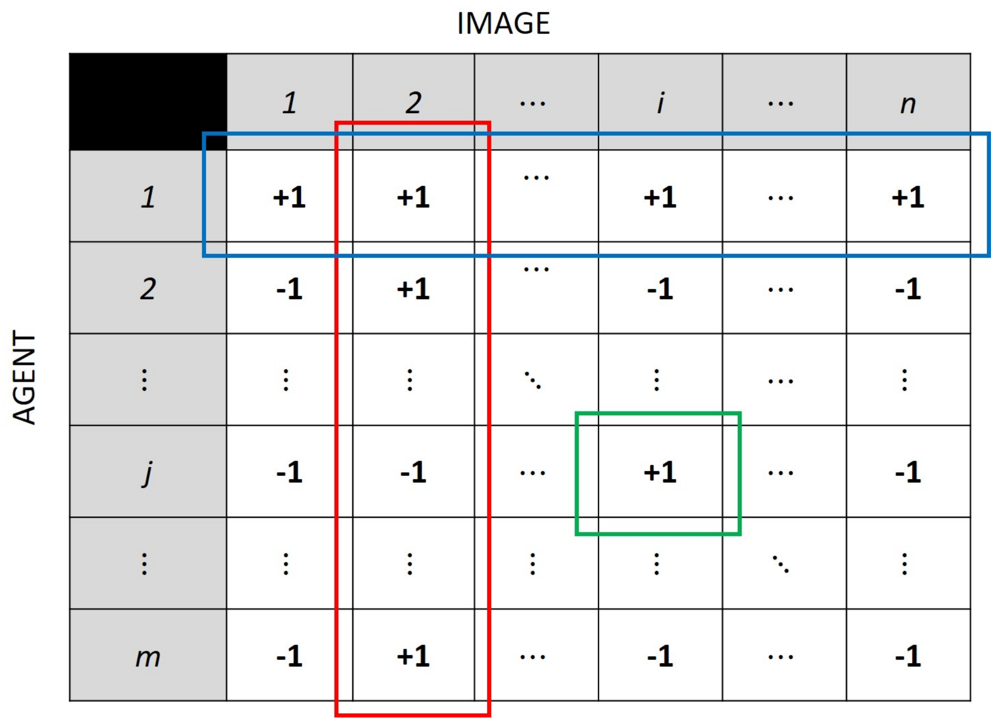

Figure 3: The classification outcome matrix, $\mathbf{A}$, captures the meta-data across all agents and tasks. The green box indicates entry A_{ji} which coincides with $\hat{y}_{ji}$ the outcome of agent j classifying task i. The red rectangle comprises the label set, $\{\hat{y}_{ji}\}_{j\in J}$, for a task, i. The blue rectangle highlights a row vector of the label set for an agent j, referred to as $\mathbf{a}_j \in \{-1,1\}^n$ in (12).

Approaches to the problem of joint classification fall into one of three categories: supervised, unsupervised, and semi-supervised. Supervised methods rely on ground-truth knowledge of a portion of the task labels in order to learn an optimal operation, f. Unsupervised methods such as our problem, typically infer meta-lables according to majority vote or averages. A third class of semi-supervised methods combine elements of both. [6, 12].

1.2 Relevant Work

Much of the related work to this problem deals with optimal crowd-sourcing, in which binary classification tasks are distributed in parallel to numerous agents at low-cost. In practice, on services such as Amazon Mechanical Turk, the responses of agents can be completely random, and it is desirable to learn a reliable binary classification for numerous tasks from the labels of noisy labelers [6]. Like our problem, it requires the assignment of binary classification tasks to labelers of unknown reliability and the need for joint classification of repeated labels from different agents for the same task.

In a very simplistic model, Karger, et al. explore the optimal crowd-sourcing problem within a framework of homogeneous agents and homogeneous tasks without supervised knowledge of any task labels. In this special case, no assignment strategy can do better than random assignment, and their approach instead focuses on a belief propagation method for inferring agent performance to perform the joint classification [7].

Ho, et al. developed an adaptive task assignment method for crowd-sourcing problems, framing the problem in very similar terms to our problem formulation [5]. They approached the assignment of heterogeneous binary tasks to heterogeneous agents within the GAP framework. Unlike in our approach, they did not constrain the joint classification to be unsupervised. Instead, they proposed an exploration-exploitation method in which additional agents were recruited to infer the difficulty of tasks, and a subset of known tasks were assigned to infer the agent accuracy, similar to a training session.

Solving GAP with the Branch and Bound algorithm (B&B) is well-established in the literature [8, 10]. We followed the Lagrangian Relaxation (LR) implementation of Fisher, et al. in [3, 4].

The problem of fully unsupervised joint classification highlights the particular benefits of the Spectral Meta-Learner (SML) introduced by Parisi, et al. [12]. SML provides a fully unsupervised method for not only joint classification but also estimating the accuracy of agents. In [12], the method is only applied in post-hoc analysis, but in our system, we will apply it adaptively, using the estimation of agent accuracy to make further task assignments.

2 Approach

2.1 Branch and Bound Algorithm

B&B is a divide and conquer optimization approach with a bounding function, search strategy, and branching strategy. The algorithm searches branches, or sub-problems of the overall problem and calculates bounds on the sub-problems according to a heuristic method. Always, a feasible solution is maintained as a lower bound for the optimal solution. This incumbent optimal solution is compared to the upper bound of all sub-problems. For a maximization problem, if the upper bound for a given sub-problem is less than the incumbent optimal solution, then the algorithm can phantom these sub-problems and all subsidiary sub-problems. Otherwise, each sub-problem which could potentially contain the optimal solution branches into further sub-problems for search. The algorithm iterates until the solution space is exhausted, and a global optimum is found.

We implemented B&B to solve GAP for the assignment module. GAP offers an intuitive branch strategy; each task assignment, $i \in I$, can possibly be assigned to any one agent, $j \in J$. This allows a systematic search though each of the n tasks and branching at each task to create m sub-problems (see Figure 4). For each explored sub-problem, if the upper bound is greater than the incumbent solution, the sub-problem and its upper bound are stored in a running queue. The search strategy consists of selecting the candidate sub-problem from the running queue with the greatest upper bound. For the bounding function, we will use LR of the semi-assignment constraints [3, 4]. B&B pseudo-code is shown in Algorithm 1 [2, 4].

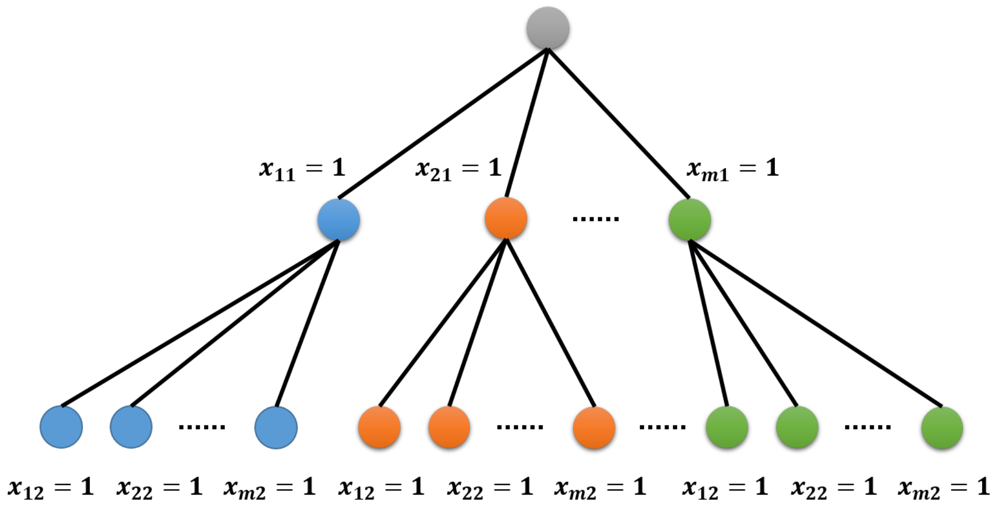

Figure 4: The B&B Algorithm for GAP. The initial candidate problem has no tasks assigned (the grey node). From this problem, the first task can be assigned to each of the m agents, creating m branches. Likewise, for tasks $i = 2, \dots, n$, each task can be assigned to one of the m agents. For each of those sub-problems, an upper bound is calculated and stored if it exceeds the incumbent optimal feasible solution.

```
Data: Target function, $f : \{0,1\}^{m \times n} \to \mathbb{R}$; Bounding function,
      $g : \{0,1\}^{m \times k} \to [\{0,1\}^{m \times n}, \mathbb{R}],\ 0 \le k < n$
Result: $x_{opt} \in \{0,1\}^{m \times n}$; $f(x_{opt}) \in \mathbb{R}$
$incumbent = 0$; $live = \{(p_0 = \emptyset, UB(p_0) = 0)\}$;
while $live \neq \emptyset$ do
    Select best candidate sub-problem, $p = \arg\max_{p_i \in live} UB(p_i)$; $live = live \setminus \{p\}$;
    for $j = 1, \dots, m$ do
        Generate sub-problem, $p_j \in \{0,1\}^{m \times (k+1)}$, by assigning task $k+1$ to agent $j$;
        $[X(p_j), UB(p_j)] := g(p_j)$;
        if $UB(p_j) > incumbent$ then
            if $X(p_j)$ is feasible then
                $incumbent = f(X(p_j))$; $x_{opt} = X(p_j)$; go to end;
            else
                $live = live \cup \{(p_j, UB(p_j))\}$;
            end
        end
    end
end
```

Algorithm 1: Branch and Bound

If all $m \times n$ sub-problems are explored, B&B would be a strictly combinatorial search algorithm. The computational savings comes from sub-problems which are not enqueued. If the upper bound is less than the incumbent optimal solution, the problem is not enqueued for later search. For each sub-problem ($x_{ji} = 1,\ i \in I, j \in J$) which is not enqueued, $m \times (n - i)$ sub-problems do not require computation of the bounding function. This highlights the importance of a tight bound from the bounding function and also a feasible solution to help phantom sub-problems throughout B&B.

2.1.1 Lagrangian Relaxation

We used LR for the bounding function in our B%B. It has been demonstrated to provide a sufficiently tight bound for B&B for binary integer problems of our problem size [3, 4, 13].

Considering the set of constraints on GAP, we faced a choice of which constraint subset to relax. One obvious choice is to relax the integer constraint on $\mathbf{x}$ (constraint [3] in (1)). This results in a conventional linear problem:

$$L^z = \max_{\mathbf{x}} \sum_{i \in I} \sum_{j \in J} v_{ji} x_{ji} \tag{3}$$

subject to [1-2,4-5] of (1). And in fact, this is the method used by MATLAB and branch and cut algorithms. The non-integer solutions indeed bound Z, but we had yet other options such as relaxing the capacity constraints (constraint [1] in (1)). We introduced Lagrange multipliers, λ_j, to dualize the capacity constraints to yield:

$$L^c(\lambda) = \max_{\mathbf{x}} \left(\sum_{i \in I} \sum_{j \in J} v_{ji} x_{ji} + \sum_{j \in J} \lambda_j \left(b_j - \sum_{i \in I} c_{ji} x_{ji} \right) \right) \tag{4}$$

subject to [2-5] of (1). The problem has reduced constraints, but we have not made it remarkably easier to solve. Therefore, we introduced Lagrange multipliers, λ_i, to dualize the semi-assignment constraints (constraint 2 in (1)), which results in the following problem:

$$L^a(\lambda) = \max_{\mathbf{x}} \left(\sum_{i \in I} \sum_{j \in J} v_{ji} x_{ji} + \sum_{i \in I} \lambda_i \left(1 - \sum_{j \in J} x_{ji} \right) \right) \tag{5}$$

subject to [1,3-5] of (1). Notice that the problem now simplifies to m distinct 0-1 knapsack problems, which can be independently solved for fixed λ:

$$L^a_j(\lambda) = \max_{\mathbf{x}} \left(\sum_{i \in I} \left(v_{ji} - \lambda_i \right) x_{ji} \right), \ j \in J. \tag{6}$$

subject to [1,3-5] of (1). The dual Lagrangian problem with respect to (6) can be formulated as

$$Z_{Da} = \min_{\lambda} L^a(\lambda) = \min_{\lambda} \left(\sum_{j \in J} L^a_j(\lambda) + \sum_{i \in I} \lambda_i \right). \tag{7}$$

subject to [1,3-5] of (1).

Fisher et al show decreased computational cost with (5) over (4) [4]. Additionally, Morales et al prove the following inequality which shows that the bound from (5) is tighter than the bound for (4) and (3) [3]:

$$Z \leq \min L^a(\lambda) \leq \min L^c(\lambda) = \min L^z, \tag{8}$$

where Z is the solution to the primal problem, (1). Therefore, we have a problem formulation, Z_{Da}, of reduced difficulty and computational cost which provides a superior bound.

2.1.2 Subgradient Method

In order to solve Z_{Da}, we used a sub-gradient descent algorithm. $L^a(\lambda)$ (5) is a relatively tractable optimization problem with the exception of not being everywhere differentiable, which precludes the exclusive use of gradient based optimization methods [3]. This problem can be overcome with the use of sub-gradient methods. A sub-gradient of a function, f at t_0 is a vector, ν, such that

$$f(t) \leq f(t_0) + \nu(t - t_0), \ \forall \ t. \tag{9}$$

Fortunately, $L^a(\lambda)$, is sub-differentiable everywhere, and as with gradient based methods, the optimal solution to $L^a(\lambda)$ occurs when zero is a sub-gradient of $L^a(\lambda)$ [1, 3]. Thus, we can iteratively update the Lagrange multipliers along the sub-gradient to find a zero sub-gradient of $L^a(\lambda)$. For $g_i^k = 1 - \sum_j x_{ji}, \ i \in I$ is a sub-gradient of (5),

$$L^a(\lambda) = \max_{\mathbf{x}} \left(\sum_{i \in I} \sum_{j \in J} v_{ji} x_{ji} + \sum_{i \in I} \lambda_i \left(1 - \sum_{j \in J} x_{ji} \right) \right)$$

at λ_k, and we can use the following iterative step for the sub-gradient descent algorithm:

$$\lambda_i^{k+1} = \lambda_i^k - \alpha_k g_i^k. \tag{10}$$

Here, α must satisfy $\lim_{k \to \infty} \alpha_k = 0$ and $\lim_{n \to \infty} \sum_{k=1}^{n} \alpha_k = \infty$. Under these conditions of the step-size, this iterative procedure guarantees convergence [1, 3]. Our sub-gradient algorithm is shown in Algorithm 2. It implements a two-step procedure in which during each iteration, the optimal assignment, $\mathbf{x}$, is updated according to (5) for fixed λ^k, and then, the Lagrange multipliers are updated according to (10). Convergence is checked prior to each iteration for either sufficiently small gradient, $\|\mathbf{g}^k\|_2 < \epsilon$, or sufficiently small change in the target value, $|L^a(\lambda^k) - L^a(\lambda^{k-1})| < \delta$ for $\epsilon, \delta > 0$.

```
Data: $\mathbf{v}_j = (v_{j1}, \dots, v_{jn})^T, \mathbf{c}_j = (c_{j1}, \dots, c_{jn})^T, b_j, \lambda^0$
Result: $\mathbf{x}, Z$
$k = 0$;
while convergence condition is not met do
    for $j = 1, \dots, m$ do
        $[\mathbf{x}_j, Z_j] = knapsack(\mathbf{v}_j - \lambda^k, \mathbf{c}_j, b_j)$;
    end
    for $i = 1, \dots, n$ do
        $\lambda_i^{k+1} = \lambda_i^k - \alpha_k(1 - \sum_{j \in J} \mathbf{x}_{ji})$;
    end
    $k = k + 1, \ Z = \sum_j Z_j + \sum_i \lambda_i^k$;
end
```

Algorithm 2: Sub-gradient Method

2.1.3 Multiplier Adjustment Method

In [4], Fisher, et al. present another approach to solving the dual problem, Z_{Da}: the multiplier adjustment method. Similar to a steepest descent method, we search the solution space for the single canonical direction in which the smallest decrease in one of the Lagrange multipliers, λ_i, will result in an assignment for a currently unassigned task. Whereas the steps of the sub-gradient descent method often result in large jumps across the solution space–or alternatively, many small steps–these measured descent steps result in a smoother path toward the optimal solution. The multiplier adjustment method remains the standard for comparison of heuristic solution techniques for GAP [8, 10].

We followed the algorithm proposed in [4] for implementation of the multiplier adjustment method. See Algorithm 3 for pseudo-code.

```
Data: $\lambda_i = \max_2 v_{ji},\ x_{ji} = 0,\ \forall\ j \in J, i \in I$
Result: $\mathbf{x}, Z$
for $j \in J$ do
    % Assign tasks with value greater than the multiplier according to the knapsack problem
    $I_j^+ = \{i \in I | v_{ji} - \lambda_i > 0\}$;
    $[x_{j,I_j^+}, Z_j] = knapsack(\{v_{ji} - \lambda_i\}_{i \in I_j^+}, \{c_{ji}\}_{i \in I_j^+}, b_j)$;
end
while do
    % Assign unassigned tasks with a value equal to the multiplier to capable agents
    $\bar{I} = \{i \in I | \sum_{j \in J} x_{ji} = 0\}$; $I_j^0 = \{i \in \bar{I} | v_{ji} = \lambda_i\}$; $J_i^0 = \{j \in J | i \in I_j^0\}$; $\bar{b}_j = b_j - \sum_{i \in I} c_{ji} x_{ji}$;
    $\mathbf{x} = \arg\max_{\mathbf{x}} \sum_{j \in J_i^0} \sum_{i \in I_j^0} v_{ji} x_{ji}$, such that $\sum_{j \in J_i^0} x_{ji} \leq 1,\ i \in \bar{I},\ \sum_{i \in I_j^0} c_{ji} x_{ji} \leq \bar{b}_j,\ j \in J,$
    $x_{ji} \in \{0,1\},\ c_{ji}, b_{ji} \in \mathbb{Z}_+,\ j \in J_i^0,\ i \in I_j^0$;
    if $\boldsymbol{x}$ is feasible then
        return; % Assignment is optimal in GAP
    end
    for $i = \{i \in I | \sum_{j \in J} x_{ji} = 0\}$ do
        % For unassigned tasks, find the least decrease in the multiplier to assign task
        for $j \in J$ do
            $\delta_{ji} = Z_j(u) - v_{ji} + \lambda_i - knapsack(\{v_{jl} - \lambda_l\}_{l \in I | l \neq i}, \{c_{jl}\}_{l \in I | l \neq i}, b_j)$;
        end
    end
    $I^0 = \{i \in I | \sum_{j \in J} x_{ji} = 0,\ \min_2(\delta_{1i}, \ldots, \delta_{mi}) > 0\}$;
    if $I^0 = \emptyset$ then
        return; % No more optimal assignment exists
    else
        % Attempt to assign tasks with minimal decrease in multiplier required for the task
        to be assigned. If assignment is more optimal, then continue, else select another task.
        for $i^* \in I^0$ do
            $j^* = \arg\min(\delta_{1i^*}, \ldots, \delta_{mi^*})$; $x_{j^* i^*} = 1$;
            $[x_{j^*, i \in I | i \neq i^*}, Z_{j^*}] = knapsack(\{v_{j^* i} - \lambda_i\}_{i \in I | i \neq i^*}, \{c_{j^* i}\}_{i \in I | i \neq i^*}, b_{j^*})$;
            $I^0 = I^0 \setminus \{i^*\}$;
            if $\sum_{i \in I} x_{ji} \leq 1\ \forall j \in J$ then
                return to start of while loop;
            else
                revert $\mathbf{x}$ and try another $i^*$;
            end
        end
    end
end
```

Algorithm 3: Multiplier Adjustment Method

2.1.4 Knapsack Algorithm

Decomposing the GAP into simpler sub-problems which have efficient solution algorithms makes B&B feasible. Solution of the 0-1 knapsack problem makes-up the core component of both the sub-gradient method and the multiplier adjustment method, which must solve multiple knapsack problems at every iteration. Fortunately, the 0-1 knapsack problem can be solved by a pseudo-polynomial time dynamic programming algorithm. It is known as the knapsack problem because it can be easily visualized as having n items of unique weights and values with a knapsack of fixed weight capacity. The problem is to determine which items if packed will maximize the value in the knapsack. The dynamic programming approach leverages the fact that to determine whether an item is optimally packed for a given capacity, it helps to know the optimal packing list for a problem of a lower capacity.

The knapsack problem is a known NP-complete problem, and for reasonable size, it facilitates efficient solution. The complexity of the dynamic programming algorithm is linear in the number of tasks, $\mathcal{O}(nb_j)$, and only grows in complexity with respect to the order of b_j. Since b_j is of sufficiently low order for many problems, the dynamic programming algorithm provides a computationally efficient method for solving the knapsack problem. The solution algorithm is shown in Algorithm 4 [11].

```
Data: $\mathbf{v}_j = (v_{j1}, \ldots, v_{jn})^T, \mathbf{c}_j = (c_{j1}, \ldots, c_{jn})^T, b_j$
Result: $\mathbf{x}_j = (x_{j1}, \ldots, x_{jn})^T, Z_j$
$M = \{0\}^{n \times b_j},\ S = \{0\}^{n \times b_j}, \mathbf{x}_j = \{0\}^n$;
for $i = 1, \ldots, n$ do
    for $l = 1, \ldots, b_j$ do
        $M(i,l) = \max(M(i-1,j), M(i-1, j - c_j(i)) + v_j(i))$;
        if $M(i-1, j - c_j(i)) + v_j(i)) > M(i-1, j)$ then
            $S(i,l) = 1$;
        end
    end
end
for $i = n, \ldots, 1$ do
    if $S(i,K)$ then
        $x_j(i) = 1, K = K - c_j(i)$;
    end
end
$Z_j = M(n, b_j)$ ;
```

Algorithm 4: Knapsack Problem

2.1.5 Greedy Search

Efficient implementation of B&B requires a feasible solution to provide a lower bound for solutions:

$$Z_{feasible} \leq Z \leq Z_{Da}. \tag{11}$$

A tight lower bound requires fewer problems to be enqueued during iterations of the B&B. Algorithm 5 shows the greedy search procedure implemented in our B&B. The procedure is based on step (3) in [4].

```
Data: $\mathbf{v}, \mathbf{c}, \mathbf{b}$
Result: $\mathbf{x}, Z$
$\mathbf{x} = bound(\mathbf{v}, \mathbf{c}, \mathbf{b})$;
if $\boldsymbol{x}$ is feasible then
    $Z = \mathbf{v}^T\mathbf{x}$, return;
else
    $I_0 = \{i \in I | \sum_{j \in J} x_{ji} = 1\}$;
    for $i \in I_0$ do
        $x_{ji} = 0 \; \forall \; j \in J$;
        1. $sort(v_{ji})$
        2. assign $x_{ji} \; \forall \; j \in J, \; i \in I_0$;
        if $\boldsymbol{x}$ is feasible then
            $Z = \mathbf{v}^T\mathbf{x}$, return;
        end
    end
end
```

Algorithm 5: Greedy Search

This procedure proved to be remarkably effective (see Figure 5); however, it is entirely heuristic and not guaranteed to produce a feasible solution. It will fail to scale appropriately with larger, more difficult problems. Currently, we make a single pass to un-assign a single task and attempt to find a feasible solution. B&B begins even if this does not yield a feasible solution. For implementation as a stand-alone method for difficult problems, it would require successive iterations of un-assigning two tasks, then three, and so on, until finding a feasible solution. This would be an unprincipled and inefficient search for large problems. Rather, the greedy method works best as a seed for B&B.

2.2 Spectral Meta-Learner

Under reasonable assumptions – independent and identically distributed task classification and conditionally independent classification from agents – the work of Parisi, et al., allows us to develop an unsupervised weighting of agents according to the balanced accuracy, $\pi_j = \frac{1}{2}(\psi_j + \eta_j)$, of each of the m agents [12]. Here, ψ is sensitivity, $\psi_j = \mathbb{P}(\hat{y}_{ji} = 1 | y_i = 1)$, and η_j is specificity, $\eta_j = \mathbb{P}(\hat{y}_{ji} = -1 | y_i = -1)$.

Consider the sample covariance matrix of predicted labels over n tasks.

$$\hat{\mathbf{Q}} = \frac{1}{m-1} \sum_{j \in J} (\mathbf{a}_j - \bar{\mathbf{a}})^T (\mathbf{a}_j - \bar{\mathbf{a}}), \tag{12}$$

where a_j is the j^{th} row of $\mathbf{A}$ (See Figure 3). As shown in [12], as $n \to \infty$, the sample covariance matrix, $\hat{\mathbf{Q}} \in \mathbb{R}^{m \times m}$ will approach:

$$Q_{ji} = \lim_{n \to \infty} \hat{Q}_{ji} = \begin{cases} 1 - \mu_j^2 & i = j \\ (1 - b^2)(2\pi_i - 1)(2\pi_j - 1) & \text{o.w.} \end{cases}. \tag{13}$$

where $\mu_j = \mathbb{E}[\mathbf{a}_j]$. This implies that the covariance matrix will be nearly rank one, $\mathbf{Q} \approx \kappa \mathbf{u}\mathbf{u}^T$, and further that $u_j \propto (2\pi_j - 1)$. The entries of the principal eigenvector will be proportional to the balanced accuracy of the corresponding agent!

Although we do not have the knowledge at hand to find $\mathbf{Q}$ analytically, we can use the sample covariance matrix as an estimate and simplify the problem of finding the principal eigenvector of $\mathbf{Q}$ to finding the principal eigenvector of a well-estimated sample covariance matrix, $\hat{\mathbf{Q}}$.

Further developed in [12], in a maximum likelihood formulation, the most likely label for a task, $\bar{y}_i$, is

$$\bar{y}_i = \text{sign}\left(\sum_{j=1}^{m} \hat{y}_{ji}\left(\log\left(\frac{\psi_j \eta_j}{(1-\psi_j)(1-\eta_j)}\right) + \log\left(\frac{\psi_j(1-\psi_j)}{\eta_j(1-\eta_j)}\right)\right)\right), \tag{14}$$

and upon a Taylor series expansion of $\bar{y}_i$ about $(\psi_j, \eta_j) = (1/2, 1/2)$, we have

$$\bar{y}_i \approx \text{sign}\left(\sum_{j=1}^{m} \hat{y}_{ji} \cdot (2\pi_j - 1)\right) \approx \text{sign}\left(\sum_{j=1}^{m} \hat{y}_{ji} \cdot u_j\right). \tag{15}$$

This means that the best mapping, f, to jointly classify a task in a maximum likelihood formulation is a weighted linear combination of agent classification labels where the weights correspond to the accuracy of each agent. Moreover, the relative reliability of each agent can be inferred from $\mathbf{u}$.

3 Implementation

Building a stable iterative system will require minimizing the time complexity of the assignment and fusion modules. The time complexity of B&B can be prohibitive for an assignment problem with too many agents and images. This requires carefully selecting the optimization technique used to provide the bounding function and solve the dual problem, (7): the sub-gradient or multiplier adjustment method.

3.1 Resources

All software was developed in MATLAB R2015b. Validation of the task assignment module ran on a Windows-based laptop computer with an Intel Core i7 2.6 GHz processor and 8GB RAM. The testing will be run on a Unix-based desktop computer with 16 Intel Core i5 2.0 GHz processors and 100GB RAM.

3.2 Validation

Since the fusion module has been developed and validated during previous work, only the assignment module requires validation. Without a known polynomial time algorithm for confirming that an assignment solution is the globally optimal solution, we use the internal MATLAB mixed integer programming function (*intlinprog*()) in the Optimization Toolbox as the known solution for validation results [2]. Additionally, as the maximum target value, Z, is not necessarily unique with respect to assignments, $\mathbf{x}$, comparisons are made according to the target value.

In [4], the results of multiple implementations of B&B were compared for various GAP problems. We will adopt these problem sizes for validation. For each unique problem size, 100 unique problems were generated with randomly generated assignment values, $\mathbf{v}$, assignment costs, $\mathbf{c}$, and agent

[2]MATLAB uses B&B with the integer constraint relaxed. Additionally, before starting B&B, ten iterations of cutting plane methods are attempted in order to reduce the size of the feasible set. This is time-consuming and computationally expensive, but reduces the growth in computation time for large problems.

budgets, $\mathbf{b}$ to accommodate feasible problems [3]. A summary of validation problem sizes is shown in Table 1.

Agents (m)	Tasks (n)	Problems
3	10	100
3	20	100
5	20	100
5	520	100
10	75	100
8	100	100
12	150	100
17	200	100

Table 1: Problem sizes of GAP for validation of B&B algorithm implementation.

We have implemented three distinct methods for solving GAP. The first is the greedy method, Algorithm 5, which does not incorporate B&B. Both the sub-gradient and multiplier adjustment methods implement B&B with Z_{Da}. We cannot verify that our methods achieved the global optimal solution, but we can verify that solutions satisfy the constraints of (1). Indeed, all three methods found feasible solutions for all problems (see Table 2).

Method	Feasible Solutions
Greedy	100
Sub-gradient	100
Multiplier Adjustment	100

Table 2: Percent (out of 800 problems) of feasible solutions to GAP validation problems returned by implemented methods.

With feasibility of solutions established, we can next compare the target value, Z, of the MATLAB solution against the greedy, sub-gradient, and multiplier adjustment solutions. Here, we measured accuracy in terms of relative error in order to account for the problem size. Results are depicted in Figure 5. Maximum relative error for all three methods is about 3%, and non-zero error occurred on less than 10% of problems. MATLAB achieved the minimum error on all problems.

Beyond accuracy, we would like to compare the speed of all three methods relative to MATLAB's implementation. Primarily, we are concerned with the time complexity of the two implementations of B&B since the greedy method is not robust enough for use in the ITAS. In order to analyze time complexity, we conducted a one-way analysis of variance (ANOVA) in which the factor is problem size.

This analysis requires the assumptions that data within a given problem size is independent and identically distributed (IID), error within problem sizes is normally distributed, and variance across problem sizes is homogeneous. The first assumption is easily satisfied by our problem set-up; however, the latter require further consideration. The combinatorial nature of B&B will necessarily create a heavy-tailed effect in the data since some problems will require the algorithm to enqueue and bound many orders of magnitude more sub-problems. This effect will also result in growth of

[3]Problems generated according to: $v_{ji} \in \mathbb{R} \sim unif(0,1)$, $c_{ji} \in \mathbb{Z}_+ \sim unif(1,10)$, and $b_j \in \mathbb{Z}_+ \sim unif(\frac{n}{m}\max_i c_{ji}, \dfrac{2n}{m} \times \max_i c_{ji})$.

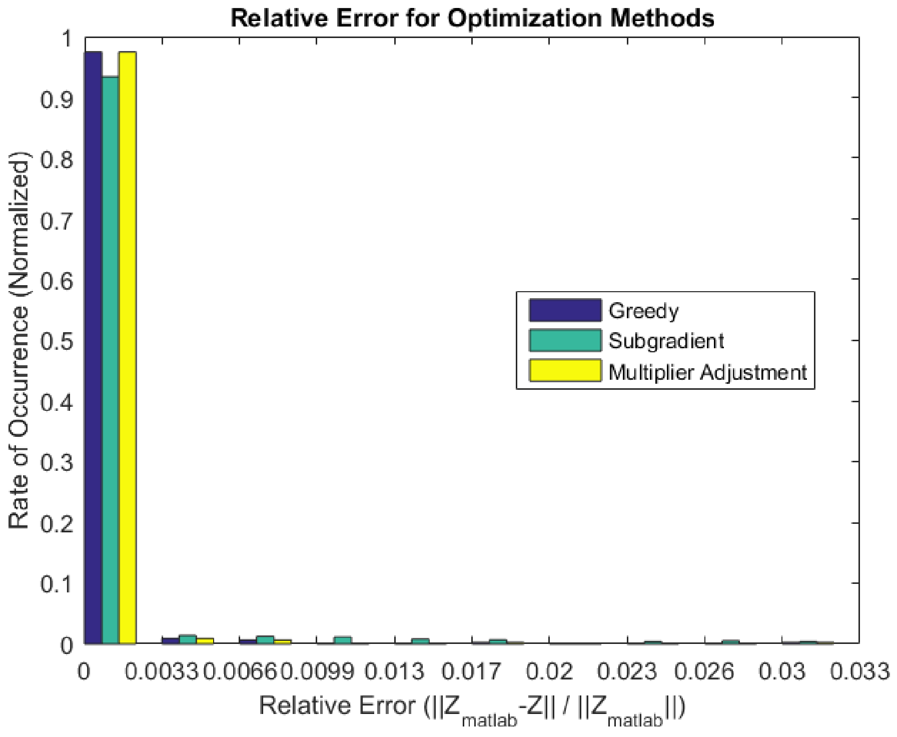

Figure 5: Histogram of the relative error of target value, Z, for greedy, sub-gradient, and multiplier adjustment methods.

variance with problem size; however, these problems will be infrequent, and if treated as outliers, the data should easily satisfy the second and third assumptions. Using a log-log scale also helped to minimize the effects of outliers.

As shown in Figure 6, the sub-gradient and multiplier adjustment methods grow linearly with respect to the problem size, $\mathcal{O}((m \times n)^{1.0})$ and $\mathcal{O}((m \times n)^{0.96})$ respectively, and MATLAB's implementation of B&B grows sub-linearly, $\mathcal{O}((m \times n)^{0.19})$, almost independent of the problem size. Note that the sub-gradient and multiplier adjustment methods compute solutions considerably faster for small problem sizes as compared to the MATLAB implementation. These differences result from the plane cutting methods implemented by MATLAB. Much of the computation time for MATLAB's algorithm is committed to ten iterations of plane cuts to reduce the size of the feasible set. For small problems, these iterations take longer than the resultant B&B but result in sub-linear time complexity over evaluated problem sizes.

Overall, these results indicate that the bulk of the time complexity of the solution of integer linear problems such as GAP, (1), results from B&B. The multiplier adjustment method solves the same dual problem, L_{Da} (7), as the sub-gradient method for each problem enqueued. In fact, the multiplier adjustment method solves that problem considerably quicker (see Figure 7); however, this is not manifested in the time complexity of the full B&B, in which the multiplier adjustment method and sub-gradient method have the same time complexity. Although MATLAB implements B&B with a similar bounding function, L^z (3), its computational time grows sub-linearly. As shown with the sub-gradient and multiplier adjustment methods, a difference in computational complexity of the bounding function will not dominate the resultant time complexity of B&B. Moreover, the bound provided by L^z is inferior to the bound provided by L^a. It must be that the reduced feasible set from which B&B searches results in the reduced time complexity.

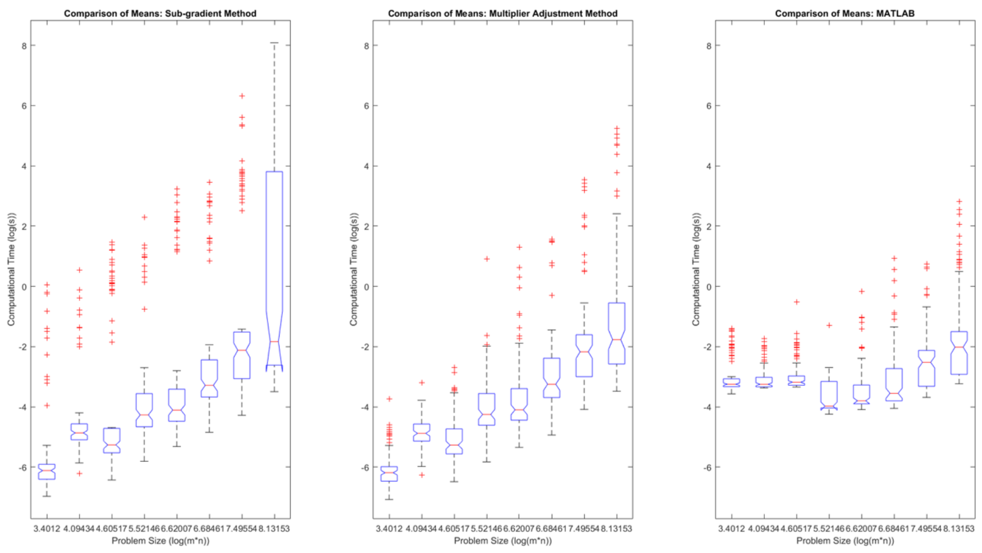

Figure 6: Problem size versus computational time ($log_e - log_e$) for solving GAP. The computation time for the sub-gradient and multiplier adjustment methods grow linearly, $\mathcal{O}((m \times n)^{1.0})$ and $\mathcal{O}((m \times n)^{0.96})$ respectively, and MATLAB's computation time grows sub-linearly, $\mathcal{O}((m \times n)^{0.19})$. The F-statistic for the three methods are 60.29, 173.8, and 49.03 respectively. The horizontal red line in the box-plots is the median of the data. The top and bottom of the box reflect the 75th and 25th percentile respectively. The red "+" represent outliers (greater than 2.7 standard deviations from the mean). Two medians are significantly different if their notch intervals do not overlap.

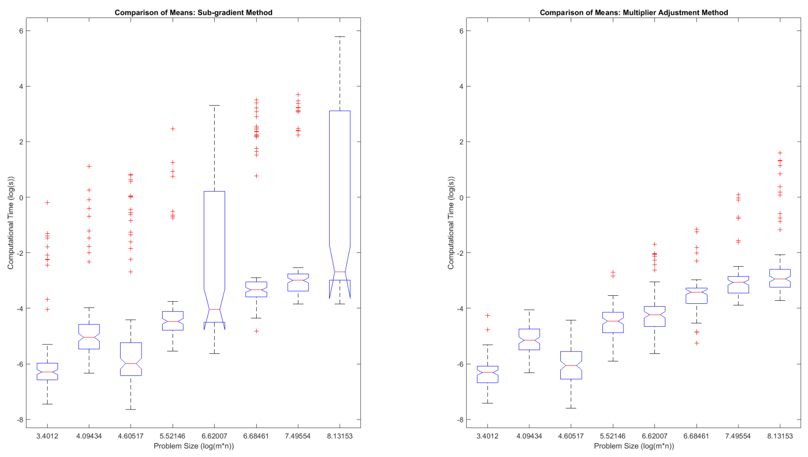

Figure 7: Problem size versus computational time ($log_e - log_e$) for solving the Dual Lagrangian, (7). The computation time for the sub-gradient method grows much faster, $\mathcal{O}((m \times n)^{0.98})$, than that of the multiplier adjustment method, $\mathcal{O}((m \times n)^{0.77})$. The F-statistic for the two methods are 58.87 and 343.3 respectively.

The difference in accuracy between the multiplier adjustment method and sub-gradient method implementations of B&B are minimal, but the time complexity of the multiplier adjustment method as a bounding function is considerably better. Additionally, the sub-gradient method enqueued significantly more problems (see Figure 8), and as a result, yielded far more outliers in computational time. These computational time outliers could be catastrophic for ITAS. Based on the validation results, for relevant problem sizes of ITAS, $\mathcal{O}(10^3)$, the multiplier adjustment method implementation of B&B will satisfy both accuracy and speed requirements.

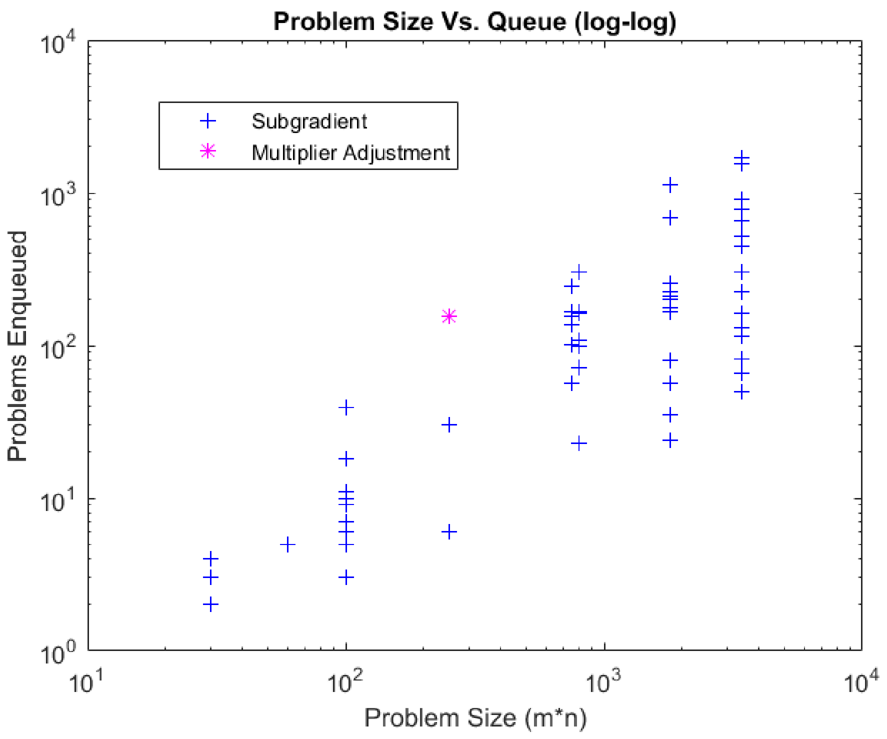

Figure 8: Problem size versus problems enqueued. The multiplier adjustment method enqueued significantly fewer sub-problems during B&B as a result of a more effective bound. The number of problems enqueued by the sub-gradient method grows linearly with problem size, $\mathcal{O}((m \times n)^{0.96})$, while the number of problems enqueued by the multiplier adjustment method is independent of problem size for the validation problems.

3.3 Testing

In order to test the system, we will use the off-line classification results of computer vision algorithms and RSVP subjects on a common image database. These off-line results will be used to simulate on-line performance of the system. Since this system should attempt to match the accuracy of assigning all images to all agents while decreasing the overall number of task assignments, we will compare the system run time and accuracy of the system versus assignment of all images to all agents.

3.4 Databases

For testing, we will use the Office Object Image Database, maintained by the Translational Neuroscience Branch of the Army Research Laboratory. The Office Object Image Database comprises a collection of images of office objects in natural and synthetic environments which pose object detection and recognition issues for both humans and computer vision. For instance, objects may be partly occluded and thus difficult for computer vision to recognize, but also objects may be un-centered in the image which makes detection during RSVP very difficult. Experimental data has been collected on RSVP subjects prompted to look for a particular office object such as a desk. Computer vision results will be collected similarly, requiring a binary response from computer vision algorithms based on the presence of a prompted office object.

3.5 Deliverables

- Software
  - Image Labeling System (fusion module, assignment module, agent classes)
  - Execution script
- Data
  - Office Object Image Database
  - Office Object RSVP Database
- Analysis
  - Performance analysis of test results
  - Implications for human-autonomous systems

3.6 Schedule

- Revise and update project plan (20 DEC - 24 JAN)
- Build Image Labeling System (25 JAN - 26 FEB)
  - Build agent classes
  - Develop message-passing framework
  - Integrate all components into a system (26 FEB)*
- Test Image Labeling System (26 FEB - 15 APR)
  - Testing (1 APR)*
  - Performance analysis of test results
- Conclusion (15 APR - 1 MAY)
  - Final Presentation and Results (6 MAY)*

(* Denotes a milestone)

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Jens Clausen. Branch and bound algorithms-principles and examples. 1999.

[3] Marshall L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 50(12_supplement):1861–1871, December 2004.

[4] Marshall L. Fisher, R. Jaikumar, and Luk N. Van Wassenhove. A Multiplier Adjustment Method for the Generalized Assignment Problem. *Management Science*, 32(9):1095–1103, September 1986.

[5] Chien-ju Ho, Shahin Jabbari, and Jennifer Wortman Vaughan. *Adaptive Task Assignment for Crowdsourced Classification.*

[6] Panagiotis G. Ipeirotis, Foster Provost, Victor S. Sheng, and Jing Wang. Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery*, 28(2):402–441, March 2013.

[7] David R. Karger, Sewoong Oh, and Devavrat Shah. Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems. *Operations Research*, 62(1):1–24, February 2014.

[8] O. Erhun Kundakcioglu and Saed Alizamir. Generalized assignment problem Generalized Assignment Problem. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1153–1162. Springer US, 2008.

[9] Jan V Leeuwen, AR Meyer, M Nival, et al. *Handbook of theoretical computer science: algorithms and complexity.* MIT Press, 1990.

[10] Dolores Romero Morales and H. Edwin Romeijn. The Generalized Assignment Problem and Extensions. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 259–311. Springer US, 2004.

[11] Ralph Otten. Lecture 13: The knapsack problem.

[12] Fabio Parisi, Francesco Strino, Boaz Nadler, and Yuval Kluger. Ranking and combining multiple predictors without labeled data. *Proceedings of the National Academy of Sciences*, 111(4):1253–1258, January 2014.

[13] G. Terry Ross and Richard M. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8(1):91–103, December 1975.

[14] P. Sajda, E. Pohlmeyer, Jun Wang, L.C. Parra, C. Christoforou, J. Dmochowski, B. Hanna, C. Bahlmann, M.K. Singh, and Shih-Fu Chang. In a Blink of an Eye and a Switch of a Transistor: Cortically Coupled Computer Vision. *Proceedings of the IEEE*, 98(3):462–478, March 2010.