Message Passing Algorithm for the Generalized Assignment Problem

Mindi Yuan, Chong Jiang, Shen Li, Wei Shen, Yannis Pavlidis, and Jun Li

Walmart Labs and University of Illinois at Urbana-Champaign
{myuan,wshen,yannis,jli1}@walmartlabs.com, {jiang17,shenli3}@illinois.edu

Abstract. The generalized assignment problem (GAP) is NP-hard. It is even APX-hard to approximate it. The best known approximation algorithm is the LP-rounding algorithm in [1] with a $(1-\frac{1}{e})$ approximation ratio. We investigate the max-product belief propagation algorithm for the GAP, which is suitable for distributed implementation. The basic algorithm passes an exponential number of real-valued messages in each iteration. We show that the algorithm can be simplified so that only a linear number of real-valued messages are passed in each iteration. In particular, the computation of the messages from machines to jobs decomposes into two knapsack problems, which are also present in each iteration of the LP-rounding algorithm. The messages can be computed in parallel at each iteration. We observe that for small instances of GAP where the optimal solution can be computed, the message passing algorithm converges to the optimal solution when it is unique. We then show how to add small deterministic perturbations to ensure the uniqueness of the optimum. Finally, we prove GAP remains strongly NP-hard even if the optimum is unique.

1 Introduction

GAP in its most general form is as follows [2]: There are multiple agents and tasks. Any agent can be assigned to perform any task with some cost or profit depending on the agent-task assignment. Each agent has a budget, and we wish to find an assignment in which no agent exceeds their budget, and the total cost of the assignment is minimized. Many practical problems can be modeled as GAP, for example finding the best locations to build distribution centers for a retail company, or assigning jobs to machines for the minimum cost in a data center. In this paper, we consider the following version of GAP:

- **Problem:** there are J jobs and M machines. Each machine has a capacity c_j. The processing cost is w_{ij} if job i is assigned to machine j.
- **Objective**: find a way to assign jobs to machines so that every job is assigned, the capacity constraints are satisfied, and the total cost is minimized.

Various algorithms have been developed for GAP. Shmoys and Tardos [3] implicitly proposed the first known algorithm, an LP-rounding 2-approximation

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 423–434, 2014.

algorithm. Subsequently, Chekuri and Khanna [4] explicitly presented that algorithm and developed a polynomial time approximation scheme for the multiple knapsack problem, which is a special case of GAP when each item has the same size and the same profit for every bin. They also proved the APX-hardness for two special cases of GAP. Recently, Fleischer *et. al.* [1] proposed two algorithms. One is a polynomial-time LP-rounding based $((1-1/e)\beta)$-approximation algorithm, which is the best known approximation for GAP so far. The other is a simple polynomial-time local search $(\beta/(\beta+1)-\epsilon)$-approximation algorithm. Cohen *et. al* [5] developed an efficient approximation algorithm, which has the same $(\beta/(\beta+1)-\epsilon)$-approximation as Fleischer's second algorithm, but is much faster.

All of the above methods are approximate. In fact, [1] showed that the results cannot be approximated within a factor better than $1-1/e$ unless $NP \in DTIME(n^{O(\log\log n)})$. However, few researchers have investigated whether better algorithms can be designed under the additional condition that the optimum is unique.

Among the message passing algorithms (MPA), belief propagation (BP) and max-product algorithms are developed corresponding to the two main problems in probabilistic inference on graphical models (GM) [6]: evaluating the marginal and maximum *a posteriori* (MAP) estimation. For loopy graphs, the correctness and convergence of BP are still open problems for arbitrary GM's. However, even for GM's with cycles, the message passing algorithms are observed to perform surprisingly well in many cases, some of which are also with rigorous proof of optimality and convergence. For example, in [7] and [8], Yuan *et. al.* proposed message passing algorithms for the minimax weight matching and the constrained assignment problem respectively. Both algorithms were proved to be correct, given uniqueness of the optimum. For the maximum weighted matching (MWM) problem, as another example, Bayati *et. al.* [9] formulated a max-product algorithm by calculating the MAP probability on a well defined GM, which encodes the data and constraints of the optimization problem. For the proof of convergence and correctness of the algorithm, they constructed an alternating path on a computation tree to show each node would choose the correct edge in a MWM after enough iterations. However, this technique does not work in our problem, where half of the nodes have a capacity constraint. In [10], Bayati *et. al.* also provided the first proof of convergence and correctness of an asynchronous BP algorithm for a combinatorial optimization. They showed that when the LP relaxation has no fractional solutions, the BP algorithm converges to the correct solution. In [11], Sanghavi showed the equivalence of LP relaxation and max-product for the weighted matching in general graphs. He provided an exact, data-dependent characterization of max-product performance, and a precise connection to LP relaxation: if the LP relaxation is tight, max-product always converges to the correct answer, and inversely, if the LP relaxation is loose, max-product does not converge.

In this paper, we propose a message passing algorithm for GAP, which computes the optimal assignment on a tree graph. The basic algorithm passes an

exponential number of real-valued messages per iteration, but a more refined version of this requires only a linear number of real-valued messages per iteration. In particular, the computation of the messages from machines to jobs decomposes into two knapsack problems, which are also present in each iteration of the LP-rounding algorithm. We observe that the algorithm can solve the GAP exactly in less than 10 iterations for small problems, when the best assignment is unique. We choose to test small problems, because their optima can be computed in reasonable amount of time. For large problems, it is hard to verify the correctness.

The rest of the paper is organized as follows. Section 2 presents the basic message passing algorithm. Section 3 derives a simplified version of the algorithm that uses fewer messages. Section 4 compares our algorithms with other algorithms. Section 5 discusses the extension of the algorithm when the optimum is not unique. Section 6 proves GAP is strongly NP-hard, even if there is a unique solution. Conclusion and future works are in Section 7.

2 Message Passing Algorithm

Consider the problem on an undirected weighted complete bipartite graph $\mathcal{G} = (\mathcal{J}, \mathcal{M}, \mathcal{E})$, where $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ denotes the n jobs and $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ denotes the m machines. Machine j $(1 \leq j \leq m)$ has a capacity c_j. Label each edge as $(J_i, M_j) \in \mathcal{E}$, with associated cost w_{ij}. The load of a machine is the sum of the weights of its adjacent edges. Assume all jobs can and need to be assigned, otherwise leaving all jobs unassigned will have the minimum cost. Although one machine can have multiple jobs, each job can only be assigned to one machine. Define an assignment matrix X, where an entry $x_{ij} = 1$ means job i is assigned to machine j and $x_{ij} = 0$ means it is not assigned to machine j. Thus the problem can be mathematically written as the following integer program:

$$\begin{aligned} \min_X & \sum_{i,j} w_{ij} x_{ij} \\ s.t. & \sum_j x_{ij} = 1, \forall i \\ & \sum_i w_{ij} x_{ij} \leq c_j, \forall j \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

Call the solution of the above problem X^*, the minimum cost assignment (MCA).

We first consider this problem on a graphical model, $\mathcal{G}$, of finding the minimum marginal distribution where the joint probability distribution can be completely specified between two nodes using the product of their functions. With abuse of notation, we will use J_i as the random variable in node J_i and M_j as the random variable for node M_j. J_i can then take on any single value l_i from $\mathcal{M}$ because each

job can only be assigned to one machine. Meanwhile, M_j can take on any *subset* $\mathcal{S}_j$ of $\mathcal{J}$, resulting in 2^n different possible values. Denote the joint probability distribution $p(J_1 = l_1, J_2 = l_2, ...J_n = l_n, M_1 = \mathcal{S}_1, M_2 = \mathcal{S}_2, ...M_m = \mathcal{S}_m)$ as

$$p(J, M) = C \prod_{i,j} \phi_{J_i,M_j}(l_i, \mathcal{S}_j) \prod_i \alpha_i(l_i) \prod_j \beta_j(\mathcal{S}_j)$$

where

$$\phi_{J_i,M_j}(l_i, \mathcal{S}_j) = \begin{cases} 1, & \texttt{if } l_i = j,\ J_i \in \mathcal{S}_j \\ 1, & \texttt{if } l_i \neq j,\ J_i \notin \mathcal{S}_j \\ +\infty, & \texttt{otherwise} \end{cases} \tag{1}$$

$$\alpha_i(l_i) = e^{w_{il_i}}$$

$$\beta_j(\mathcal{S}_j) = \begin{cases} e^{\sum_{q \in \mathcal{S}_j} w_{qj}}, \texttt{if } \sum_{q \in \mathcal{S}_j} w_{qj} \leq c_j \\ +\infty, \texttt{otherwise} \end{cases} \tag{2}$$

and C is a constant for normalization. According to the definition of the compatibility function (1), a necessary condition for $p(J, M)$ to be finite is that the assignment must be compatible, i.e. M_j must accept J_i if J_i chooses M_j, and M_j must not accept J_i if J_i does not choose M_j. According to (2), the other necessary condition is that the assignment must be feasible, i.e. the capacity constraint for each machine must be satisfied. These two conditions together are also sufficient for $p(J, M)$ to be finite, and in particular, $p(J, M) = Ce^{2\sum_i w_{il_i}}$. Note that when $p(J, M)$ is finite, it is a monotone function due to the positive edge weights. Let $p(J^*, M^*) = \arg\min p(J, M)$. By definition, $\{J_1 = l_1^*, J_2 = l_2^*, ...J_n = l_n^*\}$ will then be the MCA.

Define a message vector from J_i to M_j at iteration k: $M^k_{J_i \to M_j} = [m^k_{J_i \to M_j}(1), m^k_{J_i \to M_j}(2), ..., m^k_{J_i \to M_j}(2^n)]$. Likewise, define the message vector from M_j to J_i: $M^k_{M_j \to J_i} = [m^k_{M_j \to J_i}(1), m^k_{M_j \to J_i}(2), ..., m^k_{M_j \to J_i}(m)]$. Let $b^k_{J_i}$ be the belief vector for job J_i at the end of iteration k and let $a^k_{J_i}$ be job J_i's choice at that iteration, where $a^k_{J_i} = j$ means job J_i chooses machine M_j. Consequently, the standard message passing algorithm is as follows.

---

(1) Initialization:

$$M^0_{J_i \to M_j} = M^0_{M_j \to J_i} = \mathbf{0}$$

(2) At kth iteration:

$$m^k_{J_i \to M_j}(\mathcal{S}) = \min_{l \in \mathcal{M}} \phi_{J_i,M_j}(l, \mathcal{S}) \Big[\sum_{p \neq j} m^{k-1}_{M_p \to J_i}(l) + w_{il} \Big]$$

$$m^k_{M_j \to J_i}(l) = \min_{\mathcal{S} \subseteq \mathcal{F}_l} \phi_{J_i,M_j}(l, \mathcal{S}) \Big[\sum_{p \neq i} m^{k-1}_{J_p \to M_j}(\mathcal{S}) + \sum_{q \in \mathcal{S}} w_{ql} \Big]$$

where $\mathcal{F}_l$ is the set of all the feasible subset assignments to machine M_l.
(3) Beliefs at kth iteration:

$$b^k_{J_i}(l) = w_{il} + \sum_{p \in \mathcal{M}} m^k_{M_p \to J_i}(l)$$

(4) Assignment at the end of kth iteration:

$$a^k_{J_i} = \arg\min_{l \in \mathcal{M}} \{b^k_{J_i}(l)\}$$

In each iteration, every job/machine node sends and receives one message from every machine/job node. In computing its message, a node gathers the incoming messages at the last iteration from all neighboring nodes except the destination. Note the dimension of the vector $M^k_{J_i \to M_j}$ is 2^n. Similarly, in computing each entry for the vector $M^k_{M_j \to J_i}$, we potentially need to compare all 2^n subsets of $\mathcal{J}$, when the particular machine has enough capacity for the entire job set. As a result, the algorithm has exponential running time.

Most of the BP algorithms are formulated on trees, which are known as computation trees. In this paper, we use the same definition of computation trees as in [9]. Define the feasible tree assignment:

Definition 1. *A feasible tree assignment is an assignment on the computation tree, where 1) the capacity constraint of each machine is satisfied and 2) all the jobs, except the leaves, are assigned.*

Define $t^k_{J_i}(l)$, the total cost on the computation tree of node J_i after k iterations with the root choosing edge (J_i, M_l), i.e. J_i is believed to be assigned to machine M_l.

Lemma 1. *The belief of J_i at the kth iteration is $b^k_{J_i}(l) = 2t^k_{J_i}(l) + C$, where C is a constant depending on the initialization step of the algorithm.*

The proof is similar to that in [9] and is omitted here.

Remark 1. We only compute beliefs from the job side. If we do so from the machine side as $b^k_{M_j}(\mathcal{S}) = \sum_{q \in \mathcal{S}} w_{qj} + \sum_{p \in \mathcal{J}} m^k_{J_p \to M_j}(\mathcal{S})$, then when using the messages $M^k_{J_p \to M_j}$, we can not guarantee that the capacity constraints for the machines at the bottom of the computation tree are satisfied, which may lead to an infeasible tree assignment. This is since the capacity constraints are only incorporated in the messages $M^k_{M_j \to J_i}$, but not $M^k_{J_i \to M_j}$.

3 Simplified Algorithm

In this section, we will simplify the previous message passing algorithm to a pseudo-polynomial one. We first provide the resulting algorithm.

(1) Initialization:

$$\widetilde{m}^0_{J_i \to M_j} = \widetilde{m}^0_{M_j \to J_i} = 0$$

(2) At kth iteration:

$$\widetilde{m}^k_{J_i \to M_j} = w_{ij} - \min_{p \neq j} \left[\widetilde{m}^{k-1}_{M_p \to J_i} + w_{ip} \right]$$

$$\begin{aligned} \widetilde{m}^k_{M_j \to J_i} = & \min_{\{\overline{\mathcal{S}}, J_i\} \subseteq \mathcal{F}_j} \left[\sum_{p \in \overline{\mathcal{S}}} (\widetilde{m}^{k-1}_{J_p \to M_j} + w_{pj}) \right] + w_{ij} \\ & - \min_{\overline{\mathcal{S}} \subseteq \mathcal{F}_j} \left[\sum_{p \in \overline{\mathcal{S}}} (\widetilde{m}^{k-1}_{J_p \to M_j} + w_{pj}) \right] \end{aligned} \tag{3}$$

where $\overline{\mathcal{S}}$ is the set of all the jobs except J_i. Note the two minimizations are knapsack problems (see Remark 2).

(3) Beliefs at kth iteration:

$$b^k_{J_i}(l) = w_{il} + \widetilde{m}^k_{M_l \to J_i}$$

(4) Assignment at the end of kth iteration:

$$a^k_{J_i} = \arg\min_{l \in \mathcal{M}} \{ b^k_{J_i}(l) \}$$

To prove the equivalence of the two algorithms, we need the following lemma.

Lemma 2. *In the message passing algorithm, subtracting a constant from all the coordinates of any particular message vector at any iteration will not influence the final assignment of each job.*

The intuition behind this lemma is as follows: the algorithm only performs minimization over the messages, so subtracting an equal amount from all coordinates of a vector will still maintain the same ordering of the coordinates and hence produce the exactly same results. The proof is obvious and therefore omitted here.

Lemma 3. *The message passing algorithm and the simplified algorithm compute the same assignment for each job.*

Proof. First, we show that for any particular message vector, there are only two distinct values for each entry. Consider $m^k_{J_i \to M_j}(\mathcal{S})$. If $J_i \notin \mathcal{S}$, $m^k_{J_i \to M_j}(\mathcal{S}) = \min_{l \neq j} [\sum_{p \neq j} m^{k-1}_{M_p \to J_i}(l) + w_{il}]$. The minimization does not include the case when $l = j$, because in the case $J_i \notin \mathcal{S}$ and $l = j$, the compatibility function evaluates to $+\infty$, and thus cannot be the minimum. If $J_i \in \mathcal{S}$, again due to the property of the compatibility function, $m^k_{J_i \to M_j}(\mathcal{S}) = \sum_{p \neq j} m^{k-1}_{M_p \to J_i}(j) + w_{ij}$. As a result,

in both cases, $m^k_{J_i \to M_j}(\mathcal{S})$ does not depend on $\mathcal{S}$ and therefore takes on only two different values. The same results hold for the message $m^k_{M_j \to J_i}(l)$. Consequently, if we only pass the difference (a scalar, not a vector) of the two values and if the receiver knows the message source, then the receiver can still recover the entire message vector which includes this difference and 0. According to *Lemma 2*, passing this new vector is equivalent to passing the original one.

However, recovering the vector is not necessary. Now we show by induction that the update rule (3) computes the difference of the two distinct values in the original message vector at each iteration. For the first iteration, it is trivially true. Suppose it is true for the $k-1$th iteration. Then for the kth iteration,

$$\begin{aligned}
\widetilde{m}^k_{J_i \to M_j} &= \sum_{p \neq j} m^{k-1}_{M_p \to J_i}(j) + w_{ij} \\
&\quad - \min_{l \neq j} \Big[\sum_{p \neq j} m^{k-1}_{M_p \to J_i}(l) + w_{il} \Big] \\
&= w_{ij} - \min_{l \neq j} \Big[m^{k-1}_{M_l \to J_i}(l) + w_{il} \Big] \\
&= w_{ij} - \min_{l \neq j} \Big[\widetilde{m}^{k-1}_{M_l \to J_i} + w_{il} \Big]
\end{aligned}$$

Note in the deduction above, most of the messages in the kth iteration are 0, which can be removed. The equivalence of the updating rule for $\widetilde{m}^k_{M_j \to J_i}$ can be proved similarly. □

Remark 2. In computing the message $\widetilde{m}^k_{M_j \to J_i}$, we are actually solving two knapsack problems for machine M_j: There are $n-1$ items. Item p ($p \neq i$) has value $\widetilde{m}^{k-1}_{J_p \to M_j} + w_{pj}$ and size w_{pj}. The capacity of bin j for the first knapsack is $c_j - w_{ij}$ and the second c_j. There are many efficient methods for the singe-bin problem. Using the dynamic programming solution [12], we get a pseudo-polynomial algorithm for each knapsack. Further note that the first knapsack problem is a subproblem of the second, so we can get its solution while solving the second. This means that computing the message $\widetilde{m}^k_{M_j \to J_i}$ is equivalent to solving one knapsack problem.

4 Simulation Results

In this section, we will compare our algorithm, henceforth denoted MPA, with other algorithms in different scenarios. We will use the results obtained by the MATLAB integer programming function *bintprog*() as the optimal solution. We will compare our algorithm with the efficient GAP algorithm (EGA) in [5]. We do not compare with the local search algorithm in [1], because it has the same approximation ratio as EGA, but is much slower. We do not compare with the LP-rounding algorithm in [1] by simulations, since its complexity is much higher.

We show two sets of experiments for comparison with EGA. For the first set, the parameters are as follows. The capacity of each machine is $c_j = 100$. The

weights of the edges are drawn from a uniform distribution from 30 to 80. We run MPA with the number of iterations ranging from 0 to 9. The dimensions of the experiments range from 2×2 to 11×11, where $d \times d$ means d jobs are to be assigned to d machines. Each case is tested 1000 times. For the results returned by the two algorithms, we first verify if it is a feasible solution and then compare it with the optimum. Note that all the tests have feasible solutions, and so we define the *correct ratio* as the percentage of exactly correct solutions out of the 1000 tests.

The first 10 experiments are showed in Table 1. The first column indicates the number of iterations for MPA and the second row shows the dimension of the cost matrix. From the table we can see that MPA can reach an average correct ratio over 96.6% within 9 iterations. The smallest correct ratio for 9 iterations is 92.8% when the dimension is 11×11. The average correct ratio for EGA is 44.4% and the smallest correct ratio is 13.6%. When the dimension of the cost matrix is greater than 9×9, EGA can get 100% feasible solutions. However, the correctness is at most 21%.

Table 2 shows the case when the cost weights are drawn from a uniform distribution between 40 and 70. The average correct ratio is 91.7% for MPA and 42.1% for EGA, while the smallest correct ratio is 79.9% for MPA and 10.4% for EGA. The weights are closer now and the probability of two optima existing is therefore higher. Consequently, the correct ratio for MPA is lower than those in Table 1. Note for a particular dimension, the correct ratio will not always increase with the number of iterations. For instance, refer to the 7×7 case in Table 2; when the number of iterations increases from 8 to 9, the number of correct cases decreases by 1. This is because when the number of iterations is insufficient or when there are multiple optima, the decision of each job will oscillate; it is possible that the belief coordinate of the correct assignment is the largest at a particular iteration, but is no longer so at the next. For the cases where the MPA returns wrong solutions, we manually check them and find that either the number of iterations is insufficient or there are multiple optima.

To capture the key characteristics of the two algorithms, let us consider the following small example with cost matrix $W = \begin{pmatrix} 3 & 1 \\ 3 & 4 \end{pmatrix}$, where w_{ij} is the cost if J_i is assigned to M_j, and assume both machines have capacity 5. If we run EGA, it will first solve the knapsack problem for M_1. The following problem arises: both J_1 and J_2 have a cost of 3 if assigned to M_1, so the knapsack solution picks one at random. If it picks J_1, then the final assignment will not be optimal. However, our algorithm takes a more global view, and "knows" J_1 should wait for M_2.

In all of the experiments above, we did not change the capacity of the machines, because from the view of the jobs, it is equivalent to changing the distribution of the weights. Due to space limitations, we only show the full results from two sets of parameters for the uniform distribution. To summarize some other experiments, when the weights are drawn from uniform distributions with parameters [0, 100], [10, 90] and [20, 80], and the problem dimension is 11×11, MPA achieves nearly 100% correctness at the first iteration. In those cases, even if each job greedily chooses their least-cost machines, the capacity constraint can

Table 1. Machine capacity = 100; Cost weights ∼ uniform(30, 80); Each case tested 1000 times; Feasible solutions always exist. For each entry (a, b) in the table, a is the number of cases the returned solution is feasible out of 1000 tests and b the number of correct solutions among the feasible ones.

	MPA									
Iter	2×2	3×3	4×4	5×5	6×6	7×7	8×8	9×9	10×10	11×11
0	857, 857	819, 819	775, 775	730, 730	686, 686	661, 661	644, 644	604, 604	620, 620	620, 620
1	857, 857	819, 819	775, 775	730, 730	686, 686	661, 661	644, 644	604, 604	620, 620	620, 620
2	860, 860	819, 819	775, 775	730, 730	686, 686	661, 661	644, 644	604, 604	620, 620	620, 620
3	994, 994	980, 975	979, 975	966, 956	953, 945	953, 945	942, 929	927, 914	928, 910	908, 889
4	994, 994	980, 975	979, 975	966, 956	953, 945	953, 945	942, 929	927, 914	928, 910	908, 889
5	994, 994	995, 990	994, 990	979, 969	976, 965	972, 964	973, 958	960, 944	958, 938	947, 926
6	994, 994	995, 990	994, 990	979, 969	976, 965	972, 964	973, 958	960, 944	958, 938	947, 926
7	994, 994	997, 992	994, 990	981, 971	981, 970	978, 970	975, 960	962, 946	963, 942	949, 927
8	994, 994	997, 992	994, 990	981, 971	981, 970	978, 970	975, 960	962, 946	963, 942	949, 927
9	994, 994	997, 992	994, 990	981, 971	981, 970	978, 969	975, 960	962, 946	963, 943	950, 928
EGA	904, 875	950, 790	980, 678	992, 552	997, 423	998, 338	999, 246	1000, 210	1000, 187	1000, 136

still be satisfied with high probability. We did not test dimensions larger than 11×11, since the MATLAB function *bintprog*() became unusably slow.

Consequently, we observe that MPA appears to converge towards the correct assignment. If the weights are closer together, the problem becomes more difficult for both algorithms, but MPA consistently outperforms EGA.

5 Optimum Uniqueness

According to our simulations, the message passing algorithm works well when the optimum is unique, but this may not be the case in general. For example, if all the weights are integers and their values are close, then with high probability, there will be more than one optimum. One way to rectify this situation is to add a small deterministic perturbation to each entry of the cost matrix so that we can ensure each assignment has a unique value. Namely, if we use the same indices for the jobs and machines as before, we will need to account for m^n possible configurations. Let $\tilde{w}_{ij} = w_{ij} + (j-1)m^{-i}$, and $\tilde{c}_j = c_j + 1 - m^{-n}$. This can be viewed as appending to the value of an assignment the base-m representation of the assignment, i.e. adding the term $\sum_{i=1}^{n} m^{-i} J_i$, which is the base-$m$ number $0.J_1 J_2 \ldots J_n$. Recall that J_i is the machine assignment for job i. Since this additional value is in $[0, 1 - m^{-n}]$, and because the original capacities are integers, it follows that any assignment in the integer problem with weight matrix W and capacity vector c is valid if and only if the same assignment is valid in the modified, fractional problem with weight matrix $\tilde{W}$ and capacity vector $\tilde{c}$. Furthermore, the smallest gap between any two assignments is at least m^{-n}. As a result, the uniqueness of optimum is guaranteed.

Table 2. Cost weights ∼ uniform(40, 70); Other parameters are the same as in Table 1

	MPA									
Iter	2×2	3×3	4×4	5×5	6×6	7×7	8×8	9×9	10×10	11×11
0	803, 803	757, 757	709, 709	660, 660	593, 593	531, 531	511, 511	442, 442	411, 411	345, 345
1	803, 803	757, 757	709, 709	660, 660	593, 593	531, 531	511, 511	442, 442	411, 411	345, 345
2	803, 803	757, 757	709, 709	660, 660	593, 593	531, 531	511, 511	442, 442	411, 411	345, 345
3	992, 992	954, 950	964, 956	942, 934	914, 896	898, 877	882, 859	838, 812	809, 783	766, 722
4	992, 992	954, 950	964, 956	942, 934	914, 896	898, 877	882, 859	838, 812	809, 783	766, 722
5	992, 992	981, 975	986, 978	964, 955	944, 924	936, 911	934, 904	887, 857	870, 840	844, 787
6	992, 992	981, 975	986, 978	964, 955	944, 924	936, 911	934, 904	887, 857	870, 840	844, 787
7	992, 992	984, 978	986, 976	967, 958	953, 932	938, 912	938, 908	894, 863	881, 849	857, 799
8	992, 992	984, 978	986, 976	967, 958	953, 932	938, 912	938, 908	894, 863	881, 849	857, 799
9	992, 992	984, 978	987, 977	967, 958	953, 932	938, 911	937, 907	896, 865	885, 853	857, 799
EGA	900, 858	927, 743	972, 655	983, 542	994, 431	997, 321	998, 248	1000, 178	1000, 132	1000, 104

6 Strongly NP-Hardness

In this section, we will prove:

Theorem 1. *Given that there is a unique solution for the GAP, it is still impossible to develop a correct message passing algorithm which can terminate in pseudo-polynomial number of iterations, unless strongly NP-hard = weakly NP-hard.*

A description of strongly NP-hard can be found here [13]. For example, we know the single machine problem in our GAP can be solved in $O(Jc)$ time. Recall that J is the number of jobs and c the capacity of the single machine. If c is polynomial in J, the single machine problem can then be solved in $O(J^a)$, where a is some constant. This is an example of a weakly NP-hard problem. If the solution is still exponential in J even when c is polynomial in J, then it is called strongly NP-hard.

Now we are ready to prove the theorem.

Proof. 1) GAP is strongly NP-hard. GAP can be reduced from the 3-partition problem [14]. To see this, let each machine be a set in the 3-partition problem and let the jobs be the numbers to be partitioned. Further assume the capacities of the machines are *all equal* and the numbers are *very close* to each other. For example, consider this instance of the 3-partition problem. There are n (n is a multiple of 3) *negative integers* (since we need to do minimization for our GAP) with sum S, and each number is very close to S/n so that the sum of any two numbers is greater than $3S/n$ and any four is less than that. Set the number of machines to be $n/3$ and the capacity to be $3S/n$. At this point, the 3-partition problem is reduced to GAP. If the minimum assignment cost for GAP is S, there must be a feasible partition for the 3-partition problem. Note that S is the lowest possible cost we can reach, and we reach S only when we assign all of the jobs. Each machine would then have exactly 3 jobs due to the job size constraints.

If the minimum cost is not S, then there must not exist a feasible partition. GAP is therefore not easier than the 3-partition problem. Consequently, it is strongly NP-hard, since the 3-partition problem is strongly NP-hard [15].

2) GAP remains strongly NP-hard, even if there is one unique optimum. This can be shown by another reduction. Denote the GAP with a unique optimum as uGAP. By adding small deterministic perturbations to the GAP, as discussed in Section 5, GAP can be transformed to uGAP. This transformation takes $O(JM)$ time, which is polynomial in the input size. Clearly, a solution for uGAP is a solution for GAP, too. As a result, uGAP is not easier than GAP, and is therefore also strongly NP-hard.

3) It is not possible to develop a correct message passing algorithm which can terminate in pseudo-polynomial number of iterations, unless strongly NP-hard = weakly NP-hard. If we can have such an algorithm, we have a pseudo-polynomial algorithm for a strongly NP-hard problem, which would show strongly NP-hard = weakly NP-hard. □

Finally, it is easy to develop a dynamic programming algorithm for GAP that runs in $O(Jc^M)$. However, it is almost impossible to have a solution with complexity $O(JcM)$. In our simulations, nonetheless, the message passing algorithm is able to produce correct solutions within a reasonable number of iterations for cases with sizes up to 11×11.

7 Conclusion and Future Work

In this paper, we proposed a message passing algorithm for GAP, which is strongly NP-hard. The basic algorithm passes an exponential number of real-valued messages in each iteration. We showed that the algorithm can be simplified so that only a linear number of real-valued messages are passed in each iteration. Through simulations, we observed that our algorithm is better than the well-known approximation algorithm EGA, when the optimum is unique. Future work will include improving the algorithm and investigating the relationship between the message passing algorithm and the LP relaxation.

References

1. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum separable assignment problems. Math. of Operations Research 36, 416–431 (2011)
2. Wikipedia, Generalized assignment problem, `http://en.wikipedia.org/wiki/Generalized_assignment_problem`
3. Shmoys, D.B., Tardos, E.: An approximation algorithm for the generalized assignment problem. Math. Program 62, 461–474 (1993)
4. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 213–222 (2000)

5. Cohen, R., Katzir, L., Raz, D.: An efficient approximation for the generalized assignment problem. Info. Processing Letters 100, 162–166 (2006)
6. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press, USA (2009)
7. Yuan, M., Li, S., Shen, W., Pavlidis, Y.: Belief propagation for minimax weight matching. University of Illinois, Tech. Rep (2013)
8. Yuan, M., Shen, W., Li, J., Pavlidis, Y., Li, S.: Auction/belief propagation algorithms for constrained assignment problem. Walmart Labs, Tech. Rep. (2013)
9. Bayati, M., Shah, D., Sharma, M.: Max-product for maximum weight matching: convergence, correctness, and LP duality. IEEE Trans. Info. Theory 54, 1241–1251 (2008)
10. Bayati, M., Borgs, C., Chayes, J., Zecchina, R.: Belief propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions. SIAM J. Discrete Math. 25, 989–1011 (2011)
11. Sanghavi, S.: Equivalence of LP relaxation and max-product for weighted matching in general graphs. In: IEEE Info. Theory Workshop, pp. 242–247 (2007)
12. Wikipedia, Knapsack problem, `http://en.wikipedia.org/wiki/Knapsack_problem`
13. Strongly NP-hard, Website, `http://en.wikipedia.org/wiki/Strongly_NP-complete`
14. 3-partition problem, Website, `http://en.wikipedia.org/wiki/3-partition_problem`
15. Garey, M.R., Johnson, D.S.: Strong np-completeness results: Motivation, examples, and implications. Journal of the ACM 25, 499–508 (1978)