一般化相互割当問題の上界値を求める分散ラグランジュ緩和プロトコル

平山勝敏†

一般化相互割当問題は，分散問題解決における問題を一般的に記述することができる組合せ最適化（最大化）問題である．従来，この問題に対して，最適値の下界を与える実行可能解を求めるプロトコルは提案されているが，最適値の上界を求めるプロトコルは存在しない．本論文では，そのような上界値を求めるプロトコルとして $DisLRP_U$ を提案する．また，$DisLRP_U$ において，できるだけ小さい上界値を効率的に求めるための手法として κ-*sampling* と *lastsnap* を提案し，ベンチマーク問題を用いた実験によりそれらの性能を評価する．

Distributed Lagrangean Relaxation Protocol that Computes an Upper Bound for the Generalized Mutual Assignment Problem

Katsutoshi Hirayama†

The generalized mutual assignment problem (GMAP) is the combinatorial optimization (maximization) problem that can describe various distributed problem solving tasks. Recently, we have presented a communication protocol for this problem that enables the agents to find a feasible solution along with a lower bound. This paper provides a new protocol, called $DisLRP_U$, where the agents are engaged in computing an upper bound rather than searching a feasible solution. Furthermore, to efficiently seek for a smaller upper bound using $DisLRP_U$, we present two simple methods, called κ-*sampling* and *lastsnap*, and evaluate their performance empirically on benchmark instances.

1. はじめに

マルチエージェントシステムの研究分野では，分散プランニングや分散スケジューリングの問題，また近年では組合せオークションの問題など，組合せ最適化問題として定式化できる問題は多い．これらの問題に共通する特徴は，エージェントごとに地理的あるいは論理的に分散して存在する部分問題群が互いに何らかの制約を通して関連しつつ全体として1つの組合せ最適化問題を構成する点である．これらの問題を解く1つの方法に，分散している部分問題群を計算サーバに集め，適当なソルバ（solver）を用いて全体の問題を解くという集中型解法がある．しかし，そのような集中型解法には，部分問題群を集めるのに多大なコストがかかる☆，問題によってはセキュリティやプライバシの観点上望ましくない，などの欠点がある．そこで，近年，部分問題を解くためのソルバや他エージェントと通信する機能を各エージェントに与え，全体の問題をエージェント自身に解決させることを目指す分散型解法（プロトコル）の研究がさかんである．

組合せ最適化問題およびそれに類似した問題を解く従来のプロトコルは，エージェント間の情報伝達方式の違いにより大きく2種類に分けられる．1つは，部分問題自体は集めないが，エージェントが暫定的に求めた部分問題に対する結果を，系全体を制御するシステムエージェントに集め，その「指揮」のもとに各エージェントが繰り返し部分問題を解いて全体の問題に対する解を求めるというものである．例として，ジョブショップスケジューリング問題に対するプロトコル[9]，非線形計画問題に対するプロトコル[1]，サプライチェーン最適化問題に対するプロトコル[15]などがあり，これらはいずれも，ラグランジュ分解とよばれる部分問題を各エージェントが並行して解き，その暫定解をもとにシステムエージェントが，次にエージェ

† 神戸大学海事科学部
Faculty of Maritime Sciences, Kobe University

☆ このコストには，実際に部分問題を集めるコスト以外にも，ソルバを持つ計算サーバを設置し，維持管理するコストも含まれる．

ントが解くべき部分問題のパラメータを調整するという仕組みを持つ．この種のプロトコルでは，システムエージェントが系全体の情報を効率的に取得できるため，それを用いた探索を容易に実現できる反面，システムエージェントが効率面でのボトルネックになりうる，システムエージェントが系全体の情報を取得できるためセキュリティやプライバシの面で不安が残る，さらに，システムエージェントを導入し維持管理するコストがかかるなど，基本的には集中型解法と同様の欠点がある．

一方，もう 1 つは，そのようなシステムエージェントに頼ることなく，関連するエージェント間の通信のみでエージェントが必要な情報を取得しながら部分問題を繰り返し解くというものである．例として，組合せ最適化問題の一種である制約最適化問題を解く一連のプロトコル[5),10),12),13),17)]，決定問題である制約充足問題に対するプロトコル（分散制約充足アルゴリズム）がある[6),19)]．これらのプロトコルでは，系全体の情報を入手できるエージェントは存在せず，一般に，どのエージェントも，自分が関係しない部分にどのようなエージェントが存在し何をしているのかを知らない．したがってこの種のプロトコルには，先の集中型解法の欠点が基本的にはないといえる．

筆者は，一般化割当問題（GAP: Generalized Assignment Problem）をマルチエージェント環境に拡張した一般化相互割当問題（GMAP: Generalized Mutual Assignment Problem）を提案した[7)]．GAP は，オペレーションズリサーチの分野で古くから研究されている NP 困難な組合せ最適化問題で，容量制約のある複数のエージェントにジョブを割り当てる際，すべてのエージェントの容量制約を満たしながら効用和が最大（あるいはコスト和が最小）となるジョブの割当てを求めることを目的とする．GAP には非常に多くの先行研究があり，その厳密解法や発見的解法に関するサーベイ論文も多い[2),14),16),18)]．一方，GMAP では，GAP における各エージェントがジョブを持ち，それらを自分あるいは他のエージェントに割り当てる際に，エージェントの容量制約の下で系全体の効用和が最大となるジョブの割当てを求める．すなわち，GMAP を解くとは，何らかの原因でエージェントに不適切に割り当てられたジョブを，エージェントの容量制約の下で系全体にとっての効用和が最大となるように，エージェント自身が自律分散的に割り当てなおすことに相当する．

筆者は，GMAP に対して，システムエージェントを用いないエージェント間通信プロトコルである分散ラグランジュ緩和プロトコル（*DisLRP*: Distributed Lagrangean Relaxation Protocol）を提案した[7)]．*DisLRP* で各エージェントは，近傍のエージェントと互いに暫定的な解を交換しながら，元の問題のラグランジュ緩和問題とその双対問題を繰り返し解く．*DisLRP* は，最終的に最適解あるいは最適に近い実行可能解を得ると終了するが，つねに最適解が得られるという保証はない．その意味で *DisLRP* はヒューリスティック探索プロトコルの一種であり，GMAP の問題例に対して，その最適値の下界を与えるのみである☆．よって，本論文では以後，この特徴を明確にするために従来の *DisLRP* を改めて $DisLRP_L$ と呼ぶことにする．

これに対し本論文では，GMAP の各問題例に対して，その最適値の上界を与えるプロトコル $DisLRP_U$ を提案する．上界値を求める一般的な意義は次のとおりである．まず，上界値を用いることにより，ヒューリスティック探索などで得られた準最適な実行可能解の質を，問題の最適値が分からない場合でもある程度評価できるようになる．具体的には，得られた実行可能解の目的関数値を上界値で割ることにより，その実行可能解の質を「最適値に対して x%以上の目的関数値を持つ実行可能解である」と評価できる．また，上界値は，分枝限定法など木探索ベースのアルゴリズムの効率に影響する非常に有用な情報を提供する．したがって，上界値を求めるプロトコルは，将来的に分枝限定法をベースとしたプロトコルを開発するための重要な布石となり得る．

以下，本論文の 2 章では，GMAP の概要と GMAP を解くプロトコルを設計するうえで重要な役割を果たすラグランジュ緩和問題を説明する．3 章では，上界値を与えるプロトコル $DisLRP_U$ を導入する．$DisLRP_U$ は，既存のプロトコルである $DisLRP_L$ と基本部分を共有するため，本章ではまず，$DisLRP_L$ の基本動作の概略を説明し，その後，上界値を計算するために今回新たに組み込む $COLLECT_{TREE}$ というプロトコルを説明する．また，$DisLRP_U$ を用いてできるだけ小さい上界値を効率的に求めるために *κ-sampling* と *lastsnap* という手法を導入する．4 章では，この *κ-sampling* と *lastsnap* の性能をベンチマーク問題を用いた実験により評価し，5 章で本論文をまとめる．

☆ GMAP は目的関数値を最大化することを目標とする最大化問題であるため，その実行可能解は最適値の下界を与える．

2. GMAP

GMAP は複数のエージェントからなり，各エージェントが割り当てるべき有限個のジョブを持つ．系全体を見た場合，GMAP におけるエージェントは次のような整数計画問題 $\mathcal{GAP}$ を解く．

$$
\begin{aligned}
&\mathcal{GAP}\ \ (\text{decide } x_{kj},\ \forall k \in A,\ \forall j \in J):\\
&\text{max.} \sum_{k\in A}\sum_{j\in J} p_{kj}x_{kj}\\
&\text{s. t.} \sum_{k\in A} x_{kj} = 1,\ \ \forall j \in J, \qquad (1)\\
&\sum_{j\in J} w_{kj}x_{kj} \le c_k,\ \ \forall k \in A, \qquad (2)\\
&x_{kj} \in \{0,1\},\ \forall k \in A,\ \forall j \in J. \qquad (3)
\end{aligned}
$$

ここで，$A = \{1,...,n\}$ はエージェント集合，$J = \{1,...,l\}$ はジョブ集合，p_{kj} はエージェント k がジョブ j を選択した場合の効用，w_{kj} はエージェント k がジョブ j を選択する場合の資源消費量，c_k はエージェント k が消費できる資源の総量（資源許容量）である．また，x_{kj} は，エージェント k がジョブ j を選択する場合は 1，そうでない場合は 0 に設定される決定変数である．系全体の目的は，(1) 各ジョブがただ 1 つのエージェントに割り当てられ（割当て制約），(2) どのエージェントもその資源消費量の合計が資源許容量を超えない（容量制約），かつ，(3) どのジョブもエージェントに割り当てられるか割り当てられないかのどちらかである（01 制約），という制約条件のもとで割当ての効用和を最大化することであり，その最大値を最適値，また，最適値を実現する割当てを最適解という．

系全体で解く $\mathcal{GAP}$ に対し，その割当て制約を緩和したラグランジュ緩和問題 $\mathcal{LGAP}(\mu)$ は次のようになる[3]．

$$
\begin{aligned}
&\mathcal{LGAP}(\mu)\ \ (\text{decide } x_{kj},\ \forall k \in A,\ \forall j \in J):\\
&\text{max.} \sum_{k\in A}\sum_{j\in J} p_{kj}x_{kj} + \sum_{j\in J}\mu_j\left(1 - \sum_{k\in A} x_{kj}\right)\\
&\text{s. t.} \sum_{j\in J} w_{kj}x_{kj} \le c_k,\ \forall k \in A,\\
&x_{kj} \in \{0,1\},\ \forall k \in A,\ \forall j \in J.
\end{aligned}
$$

ここで μ_j は，ジョブ j（の割当て制約）に対応する実数値パラメータでジョブ j のラグランジュ乗数と呼ばれる．また，$\mu = (\mu_1, \mu_2, \ldots, \mu_l)$ はラグランジュ乗数ベクトルと呼ばれる．この $\mathcal{LGAP}(\mu)$ は，次のようなエージェントごとの部分問題の集合 $\{\mathcal{LGMP}_k(\mu) | k \in A\}$，ただし，

$$
\begin{aligned}
&\mathcal{LGMP}_k(\mu)\ \ (\text{decide } x_{kj}, \forall j \in R_k):\\
&\text{max.} \sum_{j\in R_k} p_{kj}x_{kj} + \sum_{j\in R_k}\mu_j\left(\frac{1}{|S_j|} - x_{kj}\right)\\
&\text{s. t.} \sum_{j\in R_k} w_{kj}x_{kj} \le c_k,\\
&x_{kj} \in \{0,1\},\ \forall j \in R_k,
\end{aligned}
$$

に分解することができる．なお，R_k は k および k 以外の他のエージェントが k に割り当てる可能性のあるすべてのジョブの集合，S_j はジョブ j が割り当てられる可能性のあるすべてのエージェントの集合☆で，任意の k，任意の j に対し，$j \in R_k \Leftrightarrow k \in S_j$ が成り立つ．ここで決定変数 x_{kj} の値はエージェント k が決定する（すなわち，各ジョブについてそれを選択するか否かはエージェント k が決定する）と仮定すると，この部分問題 $\mathcal{LGMP}_k(\mu)$ は，エージェント k の決定変数 x_{kj} しか含んでいないため，エージェント k のみにより単独で解決できる．

系全体の問題 $\mathcal{GAP}$ と部分問題 $\mathcal{LGMP}_k(\mu)$ の関係について次の 2 つの命題が任意の μ に対して成り立つ[7]．後述するが，これらの命題は GMAP を解くプロトコルを設計する際に非常に重要な役割を果たす．

命題 1 すべてのエージェントの $\mathcal{LGMP}_k(\mu)$ の最適値の和は，$\mathcal{GAP}$ の最適値の上界を与える．

命題 2 すべてのエージェントが $\mathcal{LGMP}_k(\mu)$ の最適解を得て，かつ，それらが割当て制約（$\sum_{k\in A} x_{kj} = 1,\ \ \forall j \in J$）をすべて満たしているとき，それらの最適解は $\mathcal{GAP}$ の最適解を構成している．

3. *DisLRP*$_U$

筆者は，文献 7) において GMAP の実行可能解を求めるプロトコル *DisLRP*$_L$ を提案した．本章では，まず，*DisLRP*$_U$ に関係する *DisLRP*$_L$ の基本動作を示した後，*DisLRP*$_U$ の詳細を説明する．また，*DisLRP*$_U$ を用いてできるだけ小さい上界値を効率的に求めるために導入する *κ-sampling*，および，*lastsnap* という手法を説明する．

3.1 *DisLRP*$_L$ の基本動作

個々のエージェントが *DisLRP*$_L$ に従って動作することにより，エージェント全体のマクロな振舞いは次のようになる．

(Stage 1) すべてのエージェントが，R_k 内のジョブ j のラグランジュ乗数 μ_j に初期値 0 を設定する．

☆ S_j が空集合となるようなジョブ j はそもそも割り当てる必要のないジョブで問題に含めなくてよい．したがって，$S_j \neq \emptyset$ と仮定してよい．

また，ラウンド数と呼ばれるカウンタの値を 1 に設定する．

(Stage 2) すべてのエージェントが，現在の μ の値のもとでそれぞれ $\mathcal{LGMP}_k(\mu)$ の最適解を求め，各自の近傍にそれを送る．ここでエージェント k の近傍とは，R_k 内のジョブの割当て制約に現れる変数を持つ（k 以外の）エージェントの集合で，形式的には $\bigcup_{j \in R_k} S_j \setminus \{k\}$ で表される．

(Stage 3) すべてのエージェントが，それぞれ近傍の最適解を受信してラウンド数の値を 1 増やした後，まず，R_k 内のジョブの割当て制約が満たされているかどうかをチェックする．その結果，すべてのエージェントにおいて，すべてのジョブの割当て制約が満たされていれば，命題 2 より $\mathcal{GAP}$ の最適解が得られているため終了する．そうでなければ，割当て制約が満たされていないジョブ j を R_k 内に持つエージェント，すなわち S_j 内の全エージェント，がラグランジュ乗数 μ_j の値を S_j 内で共通する値へ更新する．その後，すべてのエージェントが Stage 2 へ戻る．

このように Stage 1 でラグランジュ乗数を初期化した後，全エージェントは Stage 2 と Stage 3 を最適解が得られるまで反復する．この Stage 2 と Stage 3 の 1 回の反復をラウンドと呼び，その反復回数をラウンド数と呼ぶ．以下，各 Stage の動作に関して補足する．

Stage 1 および Stage 3 で，ジョブ j のラグランジュ乗数 μ_j に値を設定するのは，S_j 内のすべてのエージェントである．具体的には，S_j 内のすべてのエージェントが μ_j のコピーを持ち，共通の初期値と後述する共通の更新ルールにより，それらが S_j 内でつねに同じ値になるように維持される．

Stage 2 で，$\mathcal{LGMP}_k(\mu)$ の最適解を求めることは 0-1 ナップサック問題を解くことに相当する．0-1 ナップサック問題は一般には NP 困難であるが，動的計画法を用いた擬多項式時間アルゴリズムが存在するなど，NP 困難な問題の中でも比較的「易しい」部類に属するといわれている[3]．また，同問題に対する厳密解法の研究は近年でもさかんに行われており[11]，今後，さらに効率的な厳密解法が開発される可能性もある．なお，*DisLRP*$_L$ では，この 0-1 ナップサック問題を解く手法を規定しておらず，任意の厳密解法を利用してよい．また，エージェントごとに異なる厳密解法を利用してもよい．

Stage 3 で，個々のエージェントがすべてのエージェントの割当て制約が満たされていることをチェックするには，分散制約充足問題[19] を解く分散ブレイクアウト法[6] の終了判定手続きを用いる．ただし，*DisLRP*$_U$ では最適解ではなく最適値の上界を求めることが目的なので，必ずしもこの手続きを利用する必要はない．

Stage 3 で，ラグランジュ乗数の値を更新する際には劣勾配法[3] を用いる．具体的には，エージェント k は，R_k 内の任意のジョブ j に対して，近傍から受信した最適解での決定変数の値のもとで

$$g_j = 1 - \sum_{i \in S_j} x_{ij},$$

で計算される劣勾配 g_j と，ラウンド数に対して非増加となるステップ長 $leng$☆，および，集合 S_j のサイズを用いて，ジョブ j のラグランジュ乗数 μ_j を次のように更新する．

$$\mu_j \leftarrow \mu_j - leng \cdot \frac{g_j}{|S_j|}.$$

この更新ルールにより，S_j 内のすべてのエージェントが μ_j に対して同じ初期値を代入し，かつ，S_j の全員が各ラウンドにおいて同じステップ長を用いるならば，S_j 内のすべてのエージェントの μ_j は，毎回 S_j 内で共通する値へと更新される．

3.2 基本的なアイデア

前述のとおり，*DisLRP*$_L$ の基本動作の Stage 2 では，すべてのエージェントがそれぞれの $\mathcal{LGMP}_k(\mu)$ の最適解を求めている．このとき，当然，個々のエージェントの $\mathcal{LGMP}_k(\mu)$ の最適値が得られている．一方，先の命題 1 によれば，すべてのエージェントの $\mathcal{LGMP}_k(\mu)$ の最適値の和が $\mathcal{GAP}$ の最適値の上界を与える．したがって，上界値を求めるには，あるラウンド（の Stage 2）における個々のエージェントの $\mathcal{LGMP}_k(\mu)$ の最適値をすべて集めて総和を計算すればよい．*DisLRP*$_U$ は，*DisLRP*$_L$ の基本動作にこのような仕組みを導入したプロトコルである．

あるラウンドにおける $\mathcal{LGMP}_k(\mu)$ の最適値の総和を求める問題は，エージェントに分散管理されている静的なデータを用いて各エージェントが与えられた関数の値を計算する静的分散問題[8] の一種である．静的分散問題を解くには基本的には放送プロトコルを利用できるが，本研究では，より洗練されたデータ収集プロトコルである *COLLECT*$_{TREE}$[8] を利用する．

3.3 *COLLECT*$_{TREE}$ による上界値の計算

COLLECT$_{TREE}$ では，エージェントは生成木を保持していると仮定する．生成木とは，エージェントを

☆ すなわち，ある初期値と $0 < r \leq 1$ なる減衰率 r を用いて各ラウンドごとに $leng \leftarrow r \cdot leng$ と更新される．

頂点，2 エージェント間の近傍関係を枝とする無向グラフ $G=(V,E)$（V：頂点集合，E：枝集合，$n=|V|$，$m=|E|$）に対し，G と同じ頂点集合 V を持つ G の部分グラフで木構造のものをいう☆．一般に，生成木を用いることにより分散システム上の様々な問題が効率良く解決できる☆☆．

$COLLECT_{TREE}$ を用いてあるラウンド t における $\mathcal{LGMP}_k(\mu)$ の最適値の総和を求めるには，t 以降のラウンドの進行とともに，エージェントがそれぞれの最適値を生成木に沿って伝播させる．具体的には，$DisLRP_L$ の基本動作の Stage 2 において，エージェントが最適解を近傍に送る際に，近傍内のエージェントのうち生成木の枝でつながっているエージェントに対しては最適値も同時に送る．これを受けたエージェントは $DisLRP_L$ における通常の Stage 3 の動作に加え，本章で説明する $COLLECT_{TREE}$ による動作もあわせて行う．なお，ラウンド t における上界値を求めるためになされるエージェント間の一連のやりとりを本論文では以降，セッション t と呼ぶ．セッション t に関わる個々のエージェント k は，主な変数として次のものを保持しながら以下に示すように動作する．

T_k：k の近傍のうち生成木の枝でつながっているエージェントの集合

$UBL^{(t)}$：ラウンド t における個々のエージェントの最適値の集合

$DAT^{(t)}$：生成木上で k からある一定の距離に位置するエージェントのラウンド t における最適値の集合

$SCL^{(t)}$：k がセッション t に関するメッセージを送信する必要がない T_k 内のエージェントの集合

$RCL_k^{(t)}$：k がセッション t に関するメッセージを今後受信することのない T_k 内のエージェントの集合

ラウンド t での動作

エージェント k は，ラウンド t において，$\mathcal{LGMP}_k(\mu)$ の最適値 $o^{(t)}$ を求め，$UBL^{(t)} := \{(k,o^{(t)})\}$，$DAT^{(t)} := \emptyset$ とし，T_k 内のすべてのエージェントに $(t, UBL^{(t)})$ を送信する．**図 1** に例を示す．なお，この図では，グラフの頂点はエージェント，太い枝は生成木の枝を表し，ラウンド t における各エージェントの最適値とそれぞれのエージェントの識別子を同一視している．

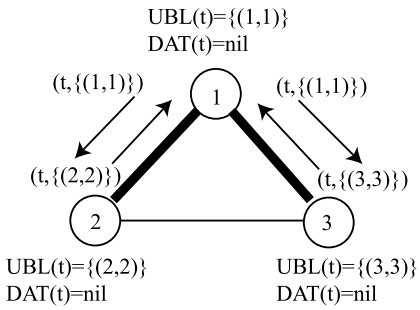


図 1 ラウンド t での動作
Fig. 1 Behavior at round t.

ラウンド $t+1$ での動作

エージェント k は，ラウンド $t+1$ において，T_k 内のあるエージェント k' から $(t, M_{k'})$ を受け取ると，$DAT^{(t)} := DAT^{(t)} \cup M_{k'}$ とする．T_k 内のすべてのエージェントからメッセージを受け取って各メッセージをこのように処理した結果，$DAT^{(t)}$ には，生成木上で k からの距離が 1 となるエージェントの最適値がすべて記録されている．

T_k 内の各エージェント k'' について，この $DAT^{(t)}$ と先にエージェント k'' から受け取ったセッション t に関するメッセージ $M_{k''}$ との差 $DAT^{(t)} \setminus M_{k''}$ を M とする．**図 2** 上側に先の例に対するここまでの計算結果を示す．

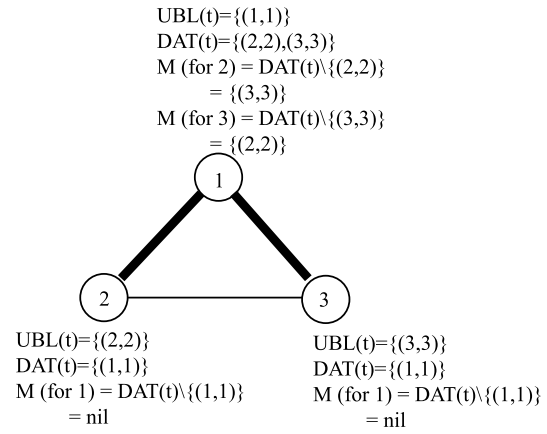
ここで，$M=\emptyset$ のとき，生成木上で k'' 以外にエージェント k からの距離が 1 であるエージェントは存在しない．したがって，次のラウンドで k から k'' に伝播させるべき他エージェントの最適値は存在しない．よってこの場合，伝播終了を意味する # を使って，エージェント k は k'' に $(t,\#)$ を送信する．

一方，$M \neq \emptyset$ のとき，生成木上で k'' 以外にもエージェント k からの距離が 1 であるエージェントが存在する．したがって，次のラウンドで k から k'' へ M を伝播させる必要があるため，エージェント k は k'' に (t,M) を送信する．

以上，T_k 内のすべてのエージェントに上記のようにそれぞれメッセージを作成して送信する．なお，ここで一度 # を送った相手には今後送信する必要はない．そこで，以降，# を送ったエージェントを順次，$SCL^{(t)}$ に追加していく．また，逆に # を受け取ったエージェントは，後述のように，それを送信したエージェントを順次，$RCL^{(t)}$ に追加していく．送信後，エージェント k は $UBL^{(t)} := UBL^{(t)} \cup DAT^{(t)}$，$DAT^{(t)} := \emptyset$ として近傍からのメッセージを待つ．前述の例に対す

☆ GMAP のような分散問題では，非連結な無向グラフの連結成分はそれぞれ独立な問題と解釈できるため，一般性を失うことなく G は連結であると仮定してよい．

☆☆ ただし，このためには生成木をあらかじめ構成しておく必要があるが，幸いにも生成木を構成する効率的な分散アルゴリズムが存在する．たとえば，Gallager-Humblet-Spira によるアルゴリズムは，メッセージ複雑度 $O(n\log n+m)$ で枝コスト付きグラフに対するコスト最小の生成木を構成できる[4]．



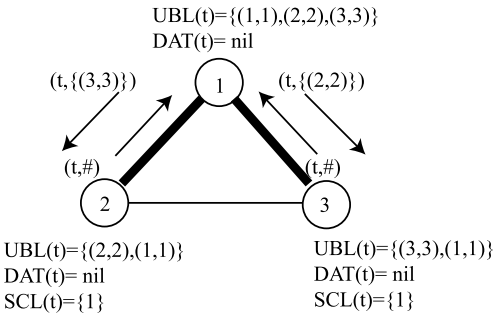


図 2 ラウンド $t+1$ での動作
Fig. 2 Behavior at round $t+1$.

るこの時点の状態を図 2 下側に示す.

ラウンド $t+t'$ $(t' \geq 2)$ における動作

エージェント k は，ラウンド $t+t'$ $(t' \geq 2)$ において，$T_k \setminus RCL^{(t)}$ 内のあるエージェント k' から $(t, M_{k'})$ を受け取ると，$M_{k'}$ の値に応じて次のように動作する.

まず，$M_{k'} = \#$ の場合，生成木上でエージェント k' を経由して k まで伝播されるべき最適値はすべて届いたことを意味する．したがって，今後，エージェント k' からセッション t に関するメッセージを待つ必要がないため，k' を $RCL^{(t)}$ に追加する．この結果，$T_k \setminus RCL^{(t)}$ が $\emptyset$ になるとすべてのエージェントの最適値が集められたことになり，それらは $UBL^{(t)}$ に記録されている．よって，それらを合計することによりエージェント k は上界値を知る.

一方，$M_{k'} \neq \#$ の場合には，$DAT^{(t)} \leftarrow DAT^{(t)} \cup M_{k'}$ とする.

$T_k \setminus RCL^{(t)}$ 内のすべてのエージェントからメッセージを受け取ってそれぞれのメッセージを上記のように処理した結果，$DAT^{(t)}$ には，生成木上で k からの距離が t' となるエージェントの最適値がすべて記録されている.

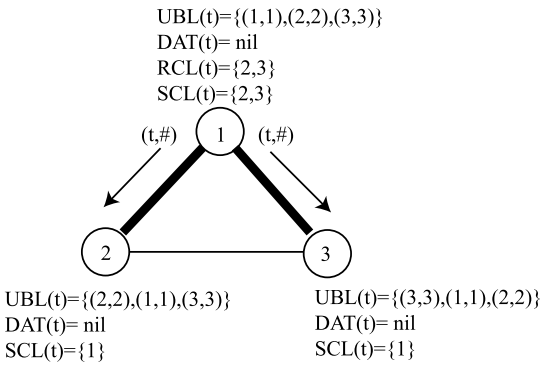


図 3 ラウンド $t+2$ での動作
Fig. 3 Behavior at round $t+2$.

$T_k \setminus SCL^{(t)}$ 内の各エージェント k'' について，この $DAT^{(t)}$ と上でエージェント k'' から受け取ったセッション t に関するメッセージ $M_{k''}$ との差 $DAT^{(t)} \setminus M_{k''}$ を M とする.

ここで，$M = \emptyset$ のとき，$M_{k''}$ に現れるエージェント以外に生成木上でエージェント k からの距離が t' であるエージェントは存在しない．したがって，次のラウンドで k から k'' に伝播させるべき他エージェントの最適値は存在しない．よってこの場合，エージェント k は k'' に $(t, \#)$ を送信する．また，k'' を $SCL^{(t)}$ に追加する.

一方，$M \neq \emptyset$ のとき，生成木上で k'' 以外にもエージェント k からの距離が t' であるエージェントが存在する．したがって，次のラウンドで k から k'' へ M を伝播させる必要があるため，エージェント k は k'' に (t, M) を送信する.

以上，$T_k \setminus SCL^{(t)}$ 内のすべてのエージェントに上記のようにそれぞれメッセージを作成して送信する．送信後，エージェント k は $UBL^{(t)} := UBL^{(t)} \cup DAT^{(t)}$，$DAT^{(t)} := \emptyset$ として近傍からのメッセージを待つ．前述の例に対するこの時点の状態を**図 3** に示す.

$COLLECT_{TREE}$ では，すべてのエージェントのラウンド t における最適値は，生成木の枝をちょうど 1 回通過するので，セッション t のメッセージ複雑度は $O(n^2)$ となる．また，セッション t に費やされるラウンド数は $O(n)$ である.

3.4 より小さい上界値を求めて

あるラウンド t における上界値は前述のように $COLLECT_{TREE}$ により求められるが，上界値はできるだけ最適値に近い，すなわち，できるだけ小さい方が望ましい．そのため，すべてのラウンドで $COLLECT_{TREE}$ を実行し，計算されたすべてのラウンドにおける上界値の中から最小の値を 1 つ選

```
procedure init
1.  round_k := 1;
2.  leng := commonValue;
3.  r := commonRatio;
4.  cutOffRound := commonValue;
5.  CTSched := set of rounds to start COLLECT_TREE;
6.  T_k := subset of Neighbors that appear in the spanning tree;
7.  minUB := maxValue;
8.  OpenSesL := φ; ClosedSesL := φ;
9.  for each job j ∈ R_k do μ_j := 0 end do;
10. Jobs_k := optimal solution to knapsack problem;
11. send assign(k, round_k, Jobs_k, φ) to Neighbors;
12. WaitL := Neighbors;
```

図 4 $DisLRP_U$：init 手続き
Fig. 4 $DisLRP_U$: init procedure.

```
when k receives assign(k', round_k', Jobs_k', M_k'k) from k' do
1.  if round_k < round_k' then add this message in DeferredL;
2.  else
3.    update AgentView with Jobs_k'; WaitL := WaitL \ {k'};
4.    for each (t, M_k') ∈ M_k'k do
5.      if M_k' = # then
6.        RCL^(t) := RCL^(t) ∪ {k'};
7.        if T_k \ RCL^(t) = φ then
8.          ub^(t) := sum of optimal values in UBL^(t);
9.          if ub^(t) < minUB then minUB := ub^(t) end if;
10.         ClosedSesL := ClosedSesL ∪ {t};
11.       end if
12.     else DAT^(t) := DAT^(t) ∪ M_k'; end if
13.   end do
14.   if WaitL = φ then
15.     stop := localcomp;
16.     if stop then output minUB; terminate the procedure;
17.     else
18.       OpenSesL := OpenSesL \ ClosedSesL;
19.       WaitL := Neighbors;
20.       restore each message in DeferredL to process;
21.     end if;
22.   end if;
23. end if;
end do
```

図 5 $DisLRP_U$：メッセージ受信後の手続き
Fig. 5 $DisLRP_U$: message processing procedure.

択することが理想的である．しかし，前述のとおり，$COLLECT_{TREE}$ を実行するにはある程度のコストがかかり，各ラウンドごとに $COLLECT_{TREE}$ を起動するとなると，メッセージのサイズやエージェントがセッション管理のために使う変数の数などが増大し，望ましくない．

そこで本論文では，次のような手法を用いて，できるだけ小さな上界値を効率的に得ることを目指す．

κ-sampling κ ラウンドごとに $COLLECT_{TREE}$ を起動して上界値を求め，それらの最小値を最終的な上界値とする．ただし，κ は正の整数である．

lastsnap あらかじめ決められたラウンド数の上限である最終ラウンドでのみ $COLLECT_{TREE}$ を起動して上界値を求め，その値を最終的な上界値とする．

なお，これらの手法の効果については，4 章の実験により検証する．

```
procedure localcomp
1.  round_k := round_k + 1;
2.  if round_k > cutOffRound then return true;
3.  else
4.    leng := leng * r;
5.    for each job j ∈ R_k do
6.      calculate subgradient g_j based on agentView;
7.      if g_j ≠ 0 then μ_j := μ_j − leng * g_j/|S_j|; end if;
8.    end do;
9.    Jobs_k := optimal solution to knapsack problem;
10.   for each k'' ∈ T_k do M_kk'' := φ; end do
11.   for each session t ∈ OpenSesL do
12.     for each k'' ∈ T_k \ SCL^(t) do
13.       (t, M_k'') := element in M_k''k concerning session t;
14.       M := DAT^(t) \ M_k'';
15.       if M = φ then
16.         M_kk'' := M_kk'' ∪ {(t, #)}; SCL^(t) := SCL^(t) ∪ {k''};
17.       else M_kk'' := M_kk'' ∪ {(t, M)}; end if
18.     end do
19.     UBL^(t) := UBL^(t) ∪ DAT^(t); DAT^(t) := φ;
20.   end do
21.   if round_k ∈ CTSched then
22.     OpenSesL := OpenSesL ∪ {round_k};
23.     o^(round_k) := optimal value of knapsack problem;
24.     UBL^(round_k) := {(k, o^(round_k))};
25.     DAT^(round_k) := φ; SCL^(round_k) := φ; RCL^(round_k) := φ;
26.     for each k'' ∈ T_k do
27.       M_kk'' := M_kk'' ∪ {(round_k, UBL^(round_k))};
28.     end do
29.   end if
30.   for each k'' ∈ Neighbors do
31.     if k'' ∈ T_k then send assign(k, round_k, Jobs_k, M_kk'') to k''
32.     else send assign(k, round_k, Jobs_k, φ) to k'' end if
33.   end do
34.   return false;
35. end if;
```

図 6 $DisLRP_U$：localcomp 手続き
Fig. 6 $DisLRP_U$: localcomp procedure.

以上，$DisLRP_L$ の基本動作に $COLLECT_{TREE}$ を組み込んだプロトコルである $DisLRP_U$ の詳細を**図 4**，**図 5**，**図 6** に示す．なお，3.3 節では，あるセッション t のみが単独で実行される場合のエージェントの動作を説明したが，図 4 から図 6 に示した $DisLRP_U$ では，連続した複数のセッション（たとえば，セッション t，セッション $t+1$，$\cdots$ など）が並行して実行される場合でも問題なく動作するよう設計されている．

4. 評　　価

低コストでできるだけ小さい上界値を得ることを目的として，本論文では 3.4 節で 2 つの手法を提案した．本章では，これらの手法の性能を実験的に評価する．

実験で用いる問題例は，OR-Library にある GAP の標準的なベンチマーク問題例☆を GMAP に変換したものである．具体的には，問題例中のすべてのジョブが一度エージェントに適当に割り当てられたと仮定し，それらをエージェント自らが最適解となるように割り当てなおすという GMAP を作成する．この GMAP

☆ http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html

は，各エージェントが最初に割り当てられた各ジョブを自分を含む系内の任意のエージェントに割り当てる問題と見なすことができ，この場合，すべてのジョブ j について $S_j = A$ となる．したがって，すべてのエージェント k について $R_k = J$，すなわち，エージェント k の部分問題 $\mathcal{LGMP}_k(\mu)$ には系全体のすべてのジョブが含まれることになる．この問題は，文献 7) の「完全」な割当て/被割当て関係を持つ GMAP に相当し，文献 7) の実験によれば，このクラスの問題は，他のクラスの問題よりも $DisLRP_L$ により得られる実行可能解の質がやや低い（下界値が小さい）という傾向が見られる．

実験は，$DisLRP_U$ の動作を模擬するシミュレータを Java で作成し，その上で行う．なお，$DisLRP_U$ では各エージェントがナップサック問題を解くソルバを持つ必要があるが，本シミュレータでは，すべてのエージェントが数理計画問題を解く CPLEX8.1 をソルバとして利用する．

本実験では，$DisLRP_U$ のパラメータのうち劣勾配法におけるステップ長 *leng* の初期値（図 4 の Step 2）とステップ長の減衰率 r（図 4 の Step 3）は 1 に固定する．また，ラウンド数の上限 *cutOffRound*（図 4 の Step 4）の値は，$\{20l, 40l, 60l, 80l, 100l\}$（$l$ はジョブ数）の範囲で変化させる．

表 1 は，$cutOffRound = 100l$ で，κ-*sampling* において κ の値をそれぞれ {1, 5, 10, 20} とした場合，および，*lastsnap* を選択した場合に得られた上界値をそれぞれの問題例ごとに示している．この表の Pr.ID の欄は問題例のラベルであり，*cnnll-i* で，*nn* はエージェント数（1 桁または 2 桁の数字），*ll* はジョブ数（2 桁の数字），i は問題例の識別子を表している．また，opt の欄は各問題例の最適値を表している．なお，上界値は，各エージェントの $\mathcal{LGMP}_k(\mu)$ の最適値の和として計算されるため，実際には小数点以下の数字を含んでいる．しかし，問題例に含まれるジョブの効用はすべて正の整数値であり最適値は必ず正の整数となるため，本実験では小数点以下を切り捨てた値を上界値としている．

表 1 から分かるとおり，利用したすべての問題例において，$\kappa > 1$ の κ-*sampling*，および，*lastsnap* は，全ラウンドで $COLLECT_{TREE}$ を起動する 1-*sampling* とほぼ同等の上界値を求めている．また，GAP としては比較的小さい規模の問題例ではあるが，全般に非常にタイトな上界値が得られている．このことから，毎ラウンド $COLLECT_{TREE}$ を起動しなくても，ときどき，あるいは，最後に一度だけ $COLLECT_{TREE}$ を起動するだけでも十分な精度の上界値が得られることが分かる．

表 1 $cutOffRound = 100l$ で得られた上界値
Table 1 Obtained upper bounds when *cutOffRound* is $100l$.

	κ-*samp*					
Pr.ID	1	5	10	20	*last*	opt
c520-1	439	439	439	439	439	434
c530-1	659	659	659	659	659	656
c832-1	762	762	762	762	762	761
c840-1	944	944	944	944	944	942
c848-1	1,134	1,134	1,134	1,134	1,135	1,133
c1030-1	710	710	710	710	710	709
c1040-1	958	958	959	959	959	958
c1050-1	1,139	1,139	1,139	1,139	1,139	1,139
c1060-1	1,451	1,451	1,451	1,451	1,451	1,451

表 2 *cutOffRound* と上界値（c848-1 の場合）
Table 2 *cutOffRound* vs. upper bound for c848-1.

	$20l$	$40l$	$60l$	$80l$	$100l$
1-samp	1,137	1,134	1,134	1,134	1,134
20-samp	1,138	1,134	1,134	1,134	1,134
last	1,138	1,135	1,135	1,134	1,135

また，*cutOffRound* の値を変化させた場合に，1-*sampling*，20-*sampling*，*lastsnap* のそれぞれで得られた上界値を**表 2** に示す．なお，この表では c848-1 の結果のみを示しているが，他の問題例についても結果は同様である．今回の実験では，*cutOffRound* の値が $40l$ 以上になるとどの手法による上界値もほぼ収束した．なお，表 2 からも明らかなように，κ-*sampling* では，得られる上界値が *cutOffRound* の値に対して必ず非増加となるが，*lastsnap* では必ずしもそうはならないことに注意すべきである．

5. むすび

本論文では，$COLLECT_{TREE}$ を利用して GMAP の上界値を求める $DisLRP_U$ というプロトコルを提案した．また，できるだけ小さい上界値を効率的に求めるための方法として κ-*sampling* と *lastsnap* を提案し，ベンチマーク問題を用いた実験によりその性能を評価した．その結果，毎ラウンド $COLLECT_{TREE}$ を起動しなくても，ときどき，あるいは，最後に一度だけ $COLLECT_{TREE}$ を起動するだけでも十分な精度の上界値が得られることが分かった．

$DisLRP_U$ は，既存のプロトコルである $DisLRP_L$ の基本部分に，あるラウンドにおける全エージェントの $\mathcal{LGMP}_k(\mu)$ の最適値をすべて集める $COLLECT_{TREE}$ を組み込んだものであり，ある意味では既存技術の組合せにすぎないともいえる．しかしながら，$DisLRP_U$

は，分散環境で計算サーバやシステムエージェントを利用せずに GMAP の上界値を求めることができる初めてのプロトコルであり，分散問題解決やマルチエージェントシステムの理論や応用技術の今後の発展においてある一定の役割を果たすと思われる．

本研究における今後の課題としては，よりタイトな上界値を得るための工夫を考案すること，また，上界値を与える実行不可能解から実行可能解をつくる，いわゆるラグランジアン・ヒューリスティック[3)] を実装したプロトコルを考案することなどがあげられる．

参 考 文 献

1) Androulakis, I.P. and Reklaitis, G.V.: Approaches to asynchronous decentralized decision making, *Computers and Chemical Engineering*, Vol.23, pp.341–355 (1999).
2) Cattrysse, D.G. and Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem, *European J. of Oper. Res.*, Vol.60, pp.260–272 (1992).
3) Fisher, M.L.: The Lagrangian relaxation method for solving integer programming problems, *Management Science*, Vol.27, No.1, pp.1–18 (1981).
4) Gallager, R.G. Humblet, P.A. and Spira, P.M.: A distributed algorithm for minimum-weight spanning tree, *ACM Trans. Prog. Lang. Syst.*, Vol.5, No.1, pp.66–77 (1983).
5) Hirayama, K. and Yokoo, M.: Distributed partial constraint satisfaction problem, *Proc. 3rd CP 1997*, pp.222–236 (1997).
6) Hirayama, K. and Yokoo, M.: The distributed breakout algorithms, *Artif. Intell.*, Vol.161, No.1-2, pp.89–115 (2005).
7) 平山勝敏：一般化相互割当問題のための分散ラグランジュ緩和プロトコル，信学論 (D-I)，Vol.J88-D-I, No.9, pp.1269–1277 (2005).
8) 亀田恒彦，山下雅史：分散アルゴリズム，近代科学社 (1994).
9) Kutanoglu, E. and Wu, S.D.: On combinatorial auction and Lagrangian relaxation for distributed resource scheduling, *IIE Transactions*, Vol.31, No.9, pp.813–826 (1999).
10) Mailler, R. and Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation, *Proc. 3rd AAMAS 2004*, pp.438–445 (2004).
11) Martello, S. Pisinger, D. and Toth, P.: New trends in exact algorithms for the 0-1 knapsack problem, *European J. of Oper. Res.*, Vol.123, pp.325–332 (2000).
12) 松井俊浩，松尾啓志，岩田 彰：分散制約最適化問題の非同期分散探索における上下界の導出と学習を用いた効率改善手法，信学論 (D-I)，Vol.J88-D-I, No.8, pp.1235–1246 (2005).
13) Modi, P.J. Shen, W.M. Tambe, M. and Yokoo, M.: An asynchronous complete method for distributed constraint optimization, *Proc. 2nd AAMAS 2003*, pp.161–168 (2003).
14) Nauss, R.M.: The generalized assignment problem, *Integer Programming: Theory and Practice*, Karlof, J.K. (Ed.), pp.39–55, CRC Press (2006).
15) 西 竜志：サプライチェーンにおける分散協調型最適化技術，人工知能学会誌，Vol.19, No.5, pp.571–578 (2004).
16) Osman, I.H.: Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches, *OR Spektrum*, Vol.17, pp.211–225 (1995).
17) Petcu, A. and Faltings, B.: A scalable method for multiagent constraint optimization, *Proc. 19th IJCAI 2005*, pp.266–271 (2005).
18) Yagiura, M. and Ibaraki, T.: Generalized assignment problem, *Handbook of Approximation Algorithms and Metaheuristics*, Gonzalez, T.F. (Ed.), Chapman & Hall/CRC in the Computer & Information Science Series (2006).
19) Yokoo, M. Durfee, E.H. Ishida, T. and Kuwabara, K.: The distributed constraint satisfaction problem: formalization and algorithms, *IEEE Trans. Knowledge and Data Engineering*, Vol.10, No.5, pp.673–685 (1998).

(平成 17 年 9 月 29 日受付)
(平成 18 年 3 月 2 日採録)

平山 勝敏（正会員）

昭和 42 年生．平成 7 年大阪大学大学院基礎工学研究科博士後期課程修了．同年より神戸商船大学助手．平成 9 年より同講師．平成 11 年から 12 年にかけてカーネギーメロン大学ロボティクス研究所客員研究員（文部省在外研究員）．平成 13 年より神戸商船大学助教授．平成 15 年より，神戸大学と神戸商船大学の統合にともない，神戸大学海事科学部助教授．主にマルチエージェントシステム，および，制約充足問題に関する研究に従事．博士（工学）．電子情報通信学会，人工知能学会，日本ソフトウエア科学会，日本オペレーションズリサーチ学会，AAAI 各会員．