RF-003

Action-GDL における集中処理化と情報漏洩を考慮する Junction-Tree の変形手法

Transformation of Junction-trees considering Centralization and Privacy-loss in Action-GDL

岩下和真 † 松井俊浩 † 松尾啓志 †
Kazuma Iwashita Toshihiro Matsui Hiroshi Matsuo

1. はじめに

近年，計算機ネットワークの発達により，複数の自律的なエージェントが通信網を介して情報の交換をしつつ，協調的に処理を行うマルチエージェントシステムが注目されている．例として，地理的に分散されたセンサにより電力などのモニタリングを行うセンサネットワーク，スマートグリッド，会議スケジューリング等が挙げられる [1, 3, 4, 9, 11]．マルチエージェントシステムでは，通信網や地理的な面で，分散して配置されたエージェントが協調して問題を解決する．そのため，各エージェントは近傍のエージェントとメッセージ通信を用いて最適化問題を解く必要がある．このようなマルチエージェントシステムで協調的に解決されるべき問題には，分散制約最適化問題 [3, 5, 6, 7, 10] として定式化できるものがある．分散制約最適化問題は，複数のエージェントに変数，制約，評価関数が分散して配置された問題であり，エージェント間のメッセージの交換を伴う．分散アルゴリズムとして構成される解法により，問題の大域的最適解を得る．

解法に利用可能なメッセージサイズ，計算量，メモリ使用量はエージェントの性能により制限される可能性がある．厳密解法において，これらは指数関数的に増加する可能性があるため，可能な限り増加を抑える様々な研究がされている．近年提案された，Action-GDL [10] では，問題を表現する Junction-Tree [2] を前処理により効率的なものに変形する．この変形により問題の一部を集中処理化することで，その後の処理でメッセージサイズを削減する．しかし，問題の変形により，各エージェントの持つ変数の値域や評価関数の情報が，他のエージェントに漏洩する程度が増加することが問題点として挙げられる．例えば，会議スケジューリングにおいて，自身の予定の一部を直接的に相手に渡すことはできるだけ避けたい場合がある．そこで本研究では，前処理である Junction-Tree の変形に新たな条件を加え，メッセージサイズの削減と情報の漏洩の抑制の両立を図る．また，Junctio-Tree が変形に適した構造を持つ場合，Action-GDL は集中処理化されやすい．しかし，Junction-Tree の構造によっては変形が十分に行えず，集中処理化が行えない場合がある．そこで，集中処理化を目指したい場合に，本来ならば Junction-Tree の変形が行えない箇所に辺を追加し，集中処理化が望まれる任意の箇所を変形するための，形式的な操作についても検討する．従来の Junction-Tree の変形手法に，エージェントの情報の漏洩を抑制する提案手法を適用し，その有用性を評価した．以降の論文の構成は次の通りである．まず，2 節では分散制約最適化問題について述べる．次に 3 節では Junction-Tree について述べる．4 節では，Action-GDL について述べ，この節で Junction-Tree の変形についても述べる．5 節で本研究の提案手法を示し，6 節で提案手法の実験と評価，提案手法の効果について述べる．最後に 7 節で本論文のまとめと今後の研究課題を述べる．

†名古屋工業大学, Nagoya Institute of Technology

2. 分散制約最適化問題（DCOP）

分散制約最適化問題 (DCOP) [3, 5, 6, 7, 10] は，エージェントに分散して配置された変数と評価関数からなる，組み合わせ最適化問題として定義される．エージェントは他のエージェントと通信を行い，評価関数の値が最もよくなるように変数の値を決定する．DCOP は $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ の要素から構成される．$\mathcal{X} = \{x_1, ..., x_n\}$ は変数の集合を表す．$\mathcal{D} = \{\mathcal{D}_1, ..., \mathcal{D}_n\}$ は各変数の値域を表す．$\mathcal{R} = \{r_1, ..., r_t\}$ は評価関数の集合を表す．評価関数 r_i は $r_i(x_j, x_k) : \mathcal{D}_j \times \mathcal{D}_k \rightarrow \mathcal{R}$ であり，変数 x_i, x_k の値の組み合わせに関する評価値を返す．r^{ij} は 2 変数 x_i, x_j に関する評価関数を表す．ただし，r^{ij} と r^{ji} は同一である．DCOP は変数を頂点とし，評価関数を辺とするグラフ (制約網) により表される．制約網の例を図 1(a) に示す．

3.Junction-Tree (JT)

3.1.Junction-Tree の定義

Junction-Tree (JT) [2] はクリークを頂点とする木であり，DCOP を表現するために用いられる．JT は $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$ により定義される．$\mathcal{X} = \{x_1, ..., x_n\}$ は変数の集合を表す．$\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_m\}$ はクリークの集合を表す．クリークは変数の部分集合であり，$\mathcal{C}_i \subseteq \mathcal{X}$ となる．$\mathcal{S}$ はセパレータの集合であり，親子関係にあるクリーク間の辺に対応する．クリーク $\mathcal{C}_i$ と $\mathcal{C}_j$ の間のセパレータは s^{ij} で表され，$s^{ij} = \mathcal{C}_i \cap \mathcal{C}_j$ である．すなわち，s^{ij} は，クリーク $\mathcal{C}_i$ と $\mathcal{C}_j$ で共通する変数の集合となる．$\Psi = \{\Psi_1, ..., \Psi_m\}$ ポテンシャルと呼ぶ．ポテンシャルは，各クリークが評価すべき評価関数を表す．ポテンシャルは，各クリークについて定義される．$P(x_i)$ により変数 x_i の親の変数を表す．また，$PP(x_i)$ により変数 x_i の擬似親の集合を表す．ただし，擬似親とは変数 x_i との間に辺がある x_i の祖先である．このときクリーク $\mathcal{C}_i$ のポテンシャルは，$\Psi_i = \bigotimes_{x_j \in \{P(x_i)\} \cup PP(x_i)} r^{ij}$ で表せる．クリークの数と変数の個数は異なってもよい．しかし，本研究では後述するように，制約網から擬似木を介して JT を生成するため，クリークは制約網や擬似木における変数，すなわちエージェントに対応する．

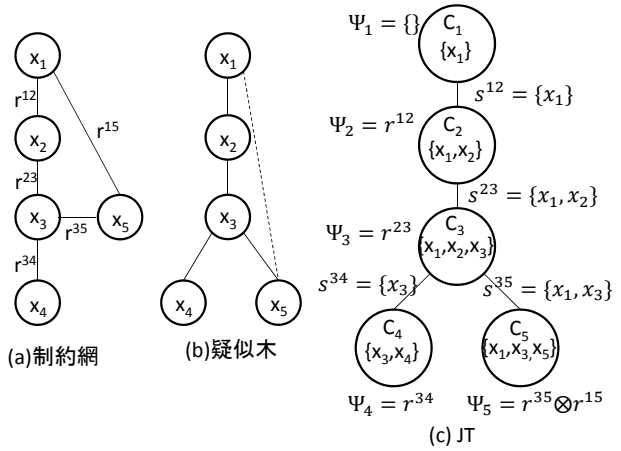

図 1: 制約網から生成した疑似木と JT の例

3.2. 擬似木の生成

JT を生成する方法はいくつかあるが，本研究では，制約網に対する擬似木から JT を生成する方法を用いる．擬似木は，制約網の生成木に基づくグラフ上の構造である．制約網に対する深さ優先探索により得られる生成木をもとに，擬似木を構成する．深さ優先探索により得られる生成木の辺を木辺と呼ぶ，また，制約網から得られる生成木に含まれない制約網の辺を後退辺と呼ぶ．図 1 では (a) が制約網を表し，(b) が (a) の制約網から生成された擬似木を表す．また，(b) の変数間における実線は木辺，破線は後退辺を表す．この例では，変数 x_1 を始点として深さ優先探索を行った．ある変数と後退辺で関係する祖先の変数を，その変数の疑似親と呼ぶ．つまり，擬似親は変数の祖先となる．

3.3.Junction-Tree の生成

疑似木に基づいて JT を生成する．JT の生成はボトムアップに行う．JT を構成するクリーク $\mathcal{C}_i$ は以下のように定義される．

$$\mathcal{C}_i = DRV(x_i) \cup IRV(x_i) \tag{1}$$

ここで，$Ch(x_i)$ により変数 x_i の子の集合を表すとき，$DRV(x_i)$ と $IRV(x_i)$ は以下のように定義される．

$$DRV(x_i) = \{x_i\} \cup \{P(x_i)\} \cup PP(x_i) \tag{2}$$

$$IRV(x_i) = \bigcup_{x_j \in Ch(x_i)} \mathcal{C}_j \setminus \{x_j\} \tag{3}$$

(2) 式は変数 x_i 自身と，親，疑似親の和集合となる．(3) 式は変数が子を持つ場合，その子に対応するクリークから，子を除いた変数の和集合となる．疑似木に基づき，すべての変数に対応するクリークを生成し，JT を構成する．図 1(c) に同図 1 の疑似木から生成した JT の例を示す．

4.Action-GDL

Action-GDL [10] は JT と動的計画法に基づいて計算を行うアルゴリズムである．前述のように，本研究では，制約網から生成した擬似木に基づく JT を対象とする．Action-GDL は JT に従って，評価値 (UTIL メッセージ‡) の伝搬と最適値の伝搬 (VALUE メッセージ) を行う．前処理として JT の変形を行うことで，より効率的な JT に基づいて問題を解くことができる．

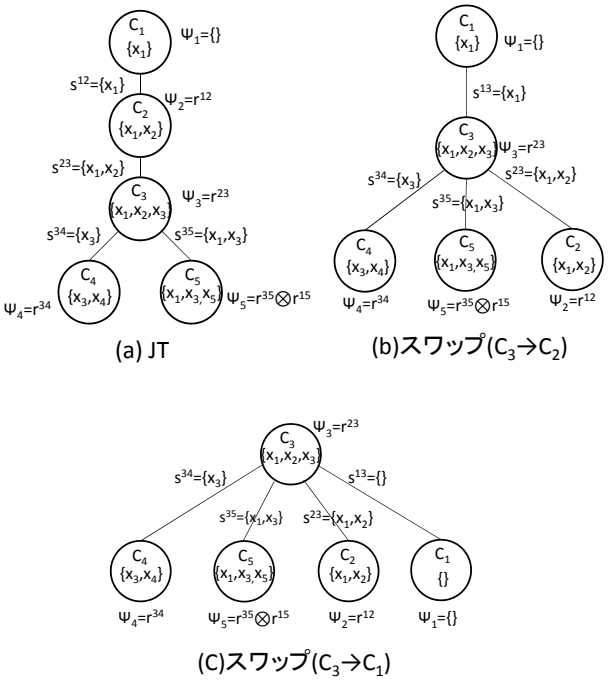

図 2: Junction-Tree の変形

4.1.Junction-Tree の変形

前処理である JT の変形について説明する．変形を行う利点は，その後の処理で UTIL メッセージサイズを削減する点である．変形はクリークのスワップを行うことで実現する．クリークのスワップはボトムアップに親子間で行う．また，スワップには条件と規則がある．子クリークを $\mathcal{C}_i$，親クリークを $\mathcal{C}_p$ とし，$\mathcal{C}_i$ と $\mathcal{C}_p$ の間でスワップを行う場合，以下の条件を満たすときに限り，親子間でスワップを行うことができる．

- $\mathcal{C}_i$ は $\mathcal{C}_p$ に対して $\mathcal{C}_p \subseteq \mathcal{C}_i$

また，スワップを行うときは，以下の規則に従って JT が変形される．

- $\mathcal{C}_p$ は $\mathcal{C}_i$ の子となる．
- $\mathcal{C}_i$ が自身に子を持つ場合，スワップ後も自身の子として引き継ぐ．
- $\mathcal{C}_p$ が子を持つ場合，スワップ後は $\mathcal{C}_i$ が $\mathcal{C}_p$ の子を自身の子として引き継ぐ．
- $\mathcal{C}_p$ が複数の子を持ち，スワップを行なえる子が複数ある場合，変数を最も多く含むクリーク間でスワップを行う．

‡UTILITY メッセージを短縮して表記する

- $\mathcal{C}_p$ はスワップ後，$Scope(\Psi_p)$ を自身のクリークに含む変数とする．ここで，Scope は引数に評価関数を取り，その評価関数を評価するために必要な変数を返す関数である．

図 2 に変形の例を示す．ボトムアップにスワップできる親子間を探すと，初めに見つかるのが $\mathcal{C}_2 \subseteq \mathcal{C}_3$ を満たす $\mathcal{C}_2$，$\mathcal{C}_3$ 間である．スワップを行うと $\mathcal{C}_2$ は $\mathcal{C}_3$ の子となる．$\mathcal{C}_2$ は $Scope(\Psi_2) = \{x_1, x_2\}$ であるため，$\mathcal{C}_2$ が含んでいる変数と同一である．そのため，スワップ後に $\mathcal{C}_2$ の含む変数は変わらない．また，このようにスワップ後，親から子となったクリークの含む変数が減少しない場合，その後の処理で UTIL メッセージサイズの削減が得られない．$\mathcal{C}_2$，$\mathcal{C}_3$ 間のスワップが (a) から (b) の変化である．次にスワップできる親子間は，$\mathcal{C}_1 \subseteq \mathcal{C}_3$ を満たす $\mathcal{C}_1$，$\mathcal{C}_3$ 間である．スワップを行うと $\mathcal{C}_1$ は $\mathcal{C}_3$ の子となる．$\mathcal{C}_1$ は $Scope(\Psi_1) = \{\}$ であるため，スワップ後に $\mathcal{C}_1$ が含む変数 x_1 がなくなり $\mathcal{C}_1$ は変数を何も含まなくなる．また，先ほどと違い，スワップ後に親から子となったクリークの含む変数が減少しているため，その後の処理で UTIL メッセージサイズの削減が得られる．$\mathcal{C}_1$，$\mathcal{C}_3$ 間のスワップが (b) から (c) の変化である．(c) の JT ではこれ以上スワップを行えないので JT の変形を終了する．

UTIL メッセージサイズの削減が得られるか否かの違いは，スワップ後に親から子となったクリークの含む変数が減少するか否かの違いによる．スワップ後に親から子となったクリークの含む変数が減少する場合，スワップ後のクリーク間におけるセパレータの要素数も減少する ((b) から (c) の変化)．このとき，セパレータの要素数は，子クリークから親クリークへ伝搬される UTIL メッセージサイズの次元となる．セパレータの要素数が減ることにより，スワップを行ったクリーク間の UTIL メッセージサイズが減る．(a) から (b) のように，スワップ後に親から子となったクリークが含む変数が減少しない場合，クリーク間のセパレータの要素数が変化しないので，UTIL メッセージサイズは削減されない．

4.2.UTIL メッセージの伝搬

UTIL メッセージの伝搬は，JT に基づいて葉から根へボトムアップに行う．クリークは自身の Ψ が評価すべき評価関数となる．UTIL メッセージを伝搬するクリークは，自身のクリークに含まれる変数についての評価値の表であるハイパーキューブを作る．その後，自身のクリークに含む変数のうち，クリーク間のセパレータの要素について表の集約 (すなわち評価値の最大値) を行う．そして集約された表を UTIL メッセージとして親に伝搬する．子クリークを持つクリークは，自身の Ψ を評価した評価値の表であるハイパーキューブと，子から伝搬されたすべての UTIL メッセージの結合を行う．そして親クリークとの間のセパレータの要素について集約を行い，その表を UTIL メッセージとする．根クリークはすべての子から UTIL メッセージを受け取ったとき，自身の Ψ を評価した評価値の表であるハイパーキューブと，子から受け取ったすべての UTIL メッセージを結合する．

4.3. 最適値の伝搬

UTIL メッセージの伝搬の後に，VALUE メッセージによる最適値の伝搬を行う．最適値の伝搬は根から葉へとトップダウンに行う．根クリークは子から受け取った UTIL メッセージと自身の評価値の表を結合した結果から，自身のクリークが含む変数の最適値を決定する．そして，すべての子クリークに対して，クリーク間のセパレータに含まれる変数の最適値を子クリークに伝搬する．最適値を伝搬された子クリークは，伝搬された最適値を用いて，自身のクリークに含む変数の内，最適値が伝搬されていない変数があれば，その変数の最適値を評価値の表から決定する．全ての葉クリークまで最適値を伝搬することにより，問題の最適解を決定する．

4.4.Junction-Tree の変形の問題点

JT の変形には問題点がある．JT の変形におけるクリークのスワップにより，評価関数の情報がそのまま他のエージェントに漏洩することがある．また，逆に，集中処理化を目指したいが，JT の構造上，十分にスワップを行えず，集中処理化が行えないことがある．以下では，それぞれの問題点について説明する．

4.4.1. エージェントの情報漏洩

情報の漏洩は JT の変形におけるクリークのスワップを行うことで起こる．スワップ後に親から子となったクリークは必ず葉クリークになるため，親クリークへの UTIL メッセージは自身の Ψ をクリークに含まれる変数で評価したものとなる．また親から子となったクリークの含む変数は，子から親となったクリークの含む変数の部分集合であるため，セパレータの要素と親から子となったクリークの含む変数は必ず同一のものとなる．そのため，Ψ を自身が含む変数で評価した評価値の表をそのまま親クリークへ伝搬する．このような伝搬は，評価関数をそのままで，直接関係しないエージェントに渡すことにつながるため，情報の漏洩とみなす．評価関数をそのままで，他のエージェントに渡すことは，その効果がなければ避けるべきである．本研究では，このような状況を，特に抑制すべき情報の漏洩の指標としてとらえる．

図 2 の (b) を例にして説明する．図 2 の (b) は $\mathcal{C}_3$ と $\mathcal{C}_2$ の間でスワップを行った結果を示している．このとき，$\mathcal{C}_2$ は $\mathcal{C}_3$ の子となり，葉クリークとなる．そして，$\mathcal{C}_2$ は変数 $\{x_1, x_2\}$ をキーとし，$\Psi_2 = r^{12}$ を計算した評価値の表を計算する．しかし，$s^{23} = \{x_1, x_2\}$ であり，$\mathcal{C}_2$ の含む変数と同一であるため，計算した表の集約を行うことができない．このとき，$\mathcal{C}_3$ へ伝搬する UTIL メッセージに含まれる評価値は，評価関数 r^{12} そのものとなる．すなわち，評価関数 r^{12} の情報が $\mathcal{C}_3$ へ漏洩することになる．これに対して，スワップを行っていない $\mathcal{C}_3$，$\mathcal{C}_4$ 間の UTIL メッセージについて考える．$\mathcal{C}_3$，$\mathcal{C}_4$ 間のセパレータ s^{34} は $\{x_3\}$ であり，$\mathcal{C}_4$ が含む変数 $\{x_3, x_4\}$ と一致していない．このとき，UTIL メッセージは，x_3 の各値について集約された評価値を含む．そのため，評価関数をそのまま渡すことはない．このよ

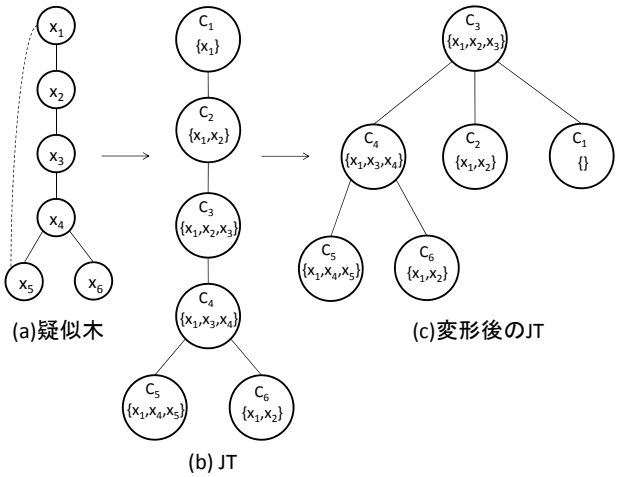

図 3: 疑似木から生成した JT と変形後の JT

うに，スワップを行った箇所でしか評価関数をそのまま渡す事態は起きない．

4.4.2. 変形の操作の制限

JT の構造上，十分にスワップを行うことができない場合がある．このような場合は，集中処理化を十分に行えない．そこで，JT の構造上，集中処理化が行えない場合について考える．ここでは，具体的に集中処理化が行えない例を示す．
図 3 では (a) の擬似木から (b) の JT を生成し，(b) の JT をクリーク C_3 とクリーク C_2 の間でスワップし，その後クリーク C_3 とクリーク C_1 の間でスワップした JT が (c) となる．集中処理化を目指したい場合，ある 1 つのクリークが他の全てのクリークを子として持つことが望ましい．しかし，スワップできる箇所は JT の構造によって制限される．図 3 における JT は，どのクリークも他の全てのクリークを子として持つことができない．このように変形の操作の制限によって，JT の集中処理化を行えない場合がある．

5. 提案手法

情報の漏洩に対する提案手法として，既存手法の JT の変形に，新たな二つの条件の追加を提案する．これにより，過剰な情報伝搬を抑制する．また，集中処理化に適さない JT に対する手法として，制約網に対する辺の追加を提案する．これにより，任意の部分問題の集中処理化を可能とする．

5.1. 過剰な情報漏洩の抑制手法

既存手法の JT の変形条件に対して，情報の漏洩とメッセージサイズを考慮した以下の二つの条件を追加する．

- スワップ回数の制限
- クリークの変数の個数を削減するスワップ

5.1.1. スワップ回数の制限

情報の漏洩はスワップを行った箇所で起こる．つまり，スワップを 1 回行う毎に情報の漏洩は起こる．そこで過剰な情報の漏洩を防ぐため，JT のすべてのクリークに対してスワップ回数を制限する．スワップ回数の上限に達しているクリークは，たとえスワップ可能な状況であってもスワップを行わない．このようにスワップ回数を制限することによって過剰な情報の漏洩を抑える．しかし，スワップ回数を制限すると，本来ならばスワップを行うことで得られたメッセージサイズの削減が得られなくなる．従って，スワップ回数をどの程度制限するかが重要となる．
スワップ回数の上限をパラメータとする場合，パラメータを大きくすると，各クリークのスワップ上限回数が多くなるため，情報の漏洩を抑制できる程度が小さくなる．しかし，UTIL メッセージサイズの削減の効果は既存手法の Action-GDL に近づく．パラメータを小さくすると，各クリークのスワップ上限回数が少なくなるため，情報の漏洩を抑制できる程度が大きくなる．しかし，UTIL メッセージサイズの削減の効果は低下する．
スワップ回数の制限の条件を形式的に表す．スワップ上限回数を max_swap_count，クリーク $\mathcal{C}_i$ が行ったスワップ回数を $\mathcal{C}_i\_swap\_count$ とするとき，条件は次式のように表される．

$$\mathcal{C}_i\_swap\_count < max\_swap\_count \tag{4}$$

5.1.2. クリークの変数の個数を削減するスワップ

クリークの変数の個数を削減するスワップとは，スワップを行う際に，親から子となったクリークの含む変数が減少するスワップのことである．スワップを行う際に，親から子となったクリークの含む変数の個数が減少する場合は，その後の処理で UTIL メッセージサイズの削減が得られる．しかし，親から子となったクリークの含んでいる変数の個数が減少しない場合は，その後の処理で UTIL メッセージのサイズが削減がされず，情報の漏洩のみが起こる．そこで，変数の個数が減少するスワップのみを行うことにより，情報の漏洩を抑制する．図 2 の (b) は，$\mathcal{C}_3$ と $\mathcal{C}_2$ の間でスワップを行った結果を示す．この箇所のクリークのスワップを見ると，親クリークである $\mathcal{C}_2$ ではスワップ前後でクリークが含む変数の個数が減少していない．従来の JT の変形ではこの箇所に対してスワップを行うが，提案手法では，この箇所に対してスワップを行わないようにし，情報の漏洩を抑える．
クリークの変数の個数を削減するスワップの条件を形式的に表す．子クリーク $\mathcal{C}_i$ と，親クリーク $\mathcal{C}_p$ の間でスワップを行うとき，条件は次式のように表される．

$$Scope(\Psi_{\mathcal{C}_p}) \subset \mathcal{C}_p \tag{5}$$

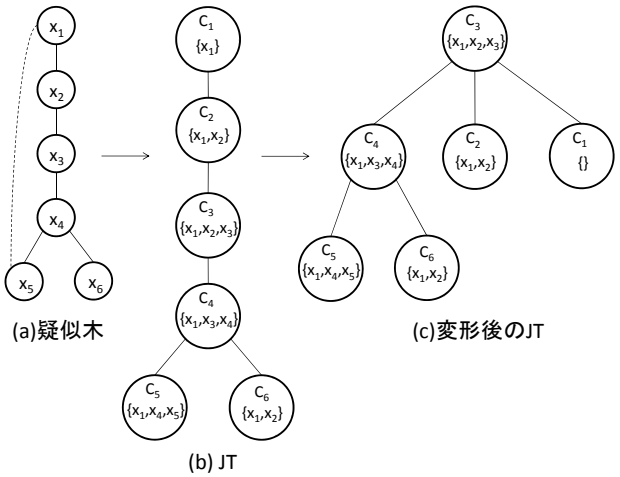

図 4: 疑似木から生成した JT と変形後の JT

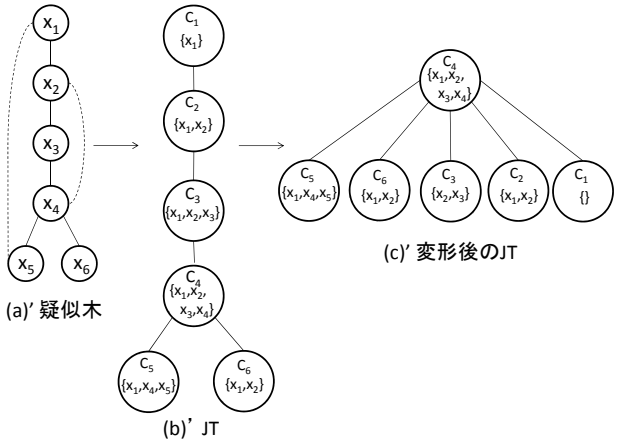

図 5: x_2 と x_4 の間に辺を追加した疑似木から生成した JT と変形後の JT

5.2. スワップの条件

上記で述べた二つの条件を用いる．以下に新たなスワップの条件を示す．子クリーク $\mathcal{C}_i$ と，親クリーク $\mathcal{C}_p$ の間でスワップを行うとき，以下の条件を同時に満たさなければならない．また，スワップを行うときは，4.1 節で示したものと同一の規則に従って JT が変形される．

- $\mathcal{C}_p \subseteq \mathcal{C}_i$
- $\mathcal{C}_i\_swap\_count < max\_swap\_count$
- $Scope(\Psi_{\mathcal{C}_p}) \subset \mathcal{C}_p$

5.3. 辺の追加による集中処理化

JT の変形では，スワップ回数が多いほど集中処理化がなされる．しかし JT の構造上，スワップを行なえる箇所が少ない場合がある．このような場合に，任意の箇所に辺を追加することで集中処理化を実現する手法を検討する．

集中処理化を実現するために辺を追加する場合，その辺に対する評価関数の値は常に 0 とする．これは，問題の最適解を失わないようにするためである．

ここでは，辺を追加する箇所について，例を用いて説明する．図 4 は図 3 と同一の問題であり，(a) の疑似木から (b) の JT を生成し，さらに (c) の JT に変形した結果を示す．集中処理化を目指したい場合，クリーク $\mathcal{C}_4$ とクリーク $\mathcal{C}_3$ の間でスワップを行えると都合が良い．クリーク $\mathcal{C}_4$ とクリーク $\mathcal{C}_3$ の間でスワップを行えば，最終的にクリーク $\mathcal{C}_4$ がすべてのクリークを自身の子として持つため，集中処理化を実現できる．例では，(b) の JT におけるクリーク $\mathcal{C}_4$ が自身の含む変数として変数 x_2 を含めば良く，(a) の疑似木における変数 x_4 と変数 x_2 の間に後退辺があれば集中処理化ができる．そこで，制約網の変数 x_4 と変数 x_2 の間に辺を加える操作を行う．変数 x_4 と変数 x_2 の間に辺を加えることは，(a) の疑似木において，変数 x_4 と変数 x_2 の間に後退辺を加えることになる．

変数 x_4 と変数 x_2 の間に辺を加えたときの疑似木，JT，変形後の JT を図 5 に示す．変数 x_4 と変数 x_2 の間へ辺を追加したため，(a)' の疑似木の変数 x_4 と変数 x_2 の間に後退辺が追加されている．この後退辺により，(a)' から生成された (b)' の JT において，クリーク $\mathcal{C}_4$ が変数 x_2 を自身のクリークに含むようになる．従って，本来ならばクリーク $\mathcal{C}_4$ とクリーク $\mathcal{C}_3$ の間でスワップを行えなかったが，辺の追加によってスワップを行なえるようになる．変形を行った JT が (c)' となり，クリーク $\mathcal{C}_4$ が他のすべてのクリークを自身の子として持つようになる．そして結局，クリーク $\mathcal{C}_4$ に対して集中処理化を行うことができる．このようにして，集中処理化を行いたい任意の箇所に辺を追加することにより，集中処理化することができる．このような辺の追加による集中処理化では，情報の漏洩が起きる一方で，メッセージサイズも増加する可能性がある．ただし，木の深さを削減することは可能である．任意の変形のための操作の一般化として，検討する意義があると考える．

6. 実験と評価

提案手法について，実験により評価を行った．比較対象として既存手法の DPOP [8]，Action-GDL を用いた．DPOP を用いた理由は，Action-GDL において JT の変形を行わずに問題を解いたとき，UTIL メッセージサイズが DPOP で解いた場合と同一であるためである．

実験では DPOP，Action-GDL，提案手法の UTIL メッセージサイズの比較と，Action-GDL と提案手法のスワップ回数を比較した．

実験に用いる問題は，次のように生成した．変数をノードと考え，まず根ノードに一様分布の乱数により 1 から 3 個の子を持つ木を生成する．木の葉ノードも同様に 1 から 3 個の子を生成する．木のノードが n 個になるまで子の生成を繰り返す．その後，任意の本数後退辺を加えることにより擬似木を生成する．さらに，擬似木に基づいて JT を生成する．

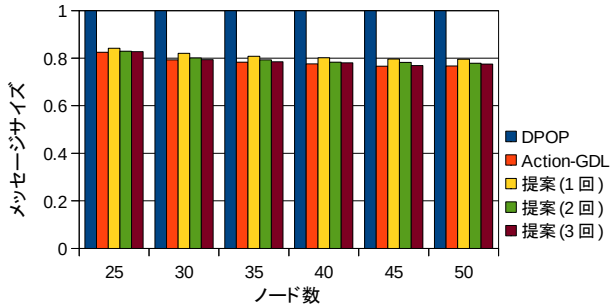

図 6: ノード数にたいする UTIL メッセージサイズ

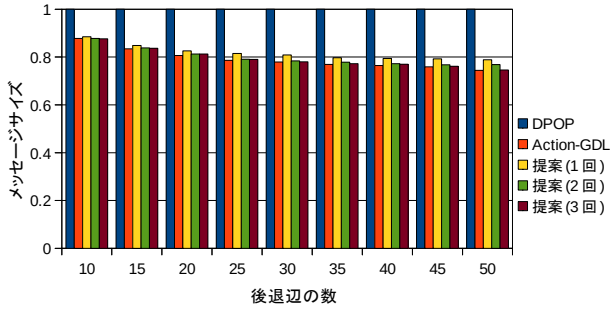

図 7: 後退辺にたいする UTIL メッセージサイズ

また，UTIL メッセージサイズは以下の式により計算した．

$$\sum_{i=1}^{n}(\prod_{x_k \in s_{ij}} |\mathcal{D}_k|)$$

情報の漏洩はスワップを行った回数として評価した．なお，いかなる JT についても根クリークに対するスワップは情報の漏洩が起こらないため，根クリークに対するスワップは情報の漏洩として数えない．

6.1. UTIL メッセージの評価

ノードの数を 40 個とし，後退辺の数を 10 本から 50 本まで，5 本ずつ増やしていった擬似木および，後退辺の数を 30 本とし，ノードの数を 25 個から 50 個まで，5 個ずつ増やしていった擬似木に基づく JT に対して，各手法の UTIL メッセージサイズを比較した．提案手法におけるスワップ回数の上限を 1 回，2 回，3 回として評価した．結果を図 6 と図 7 に示す．

これらの結果は DPOP の UTIL メッセージサイズを 1 として正規化されている．同じ後退辺の数である場合，ノード数が多くなるにつれて UTIL メッセージサイズは小さくなった．また，提案手法の UTIL メッセージサイズは既存手法の Action-GDL にそれほど劣らなかった．ノード数が同じである場合，加える後退辺の数が多くなるにつれて UTIL メッセージサイズは小さくなった．また，提案手法の UTIL メッセージサイズは既存手法の Actiuon-GDL にそれほど劣らなかった．

提案手法を上限回数の別に見ると，上限回数が 1 回の場合が最も既存手法の Action-GDL で得られる結果と差がある．これはスワップ上限回数が 1 回であるため，まだクリークのスワップが行える箇所があったとしても上限によりスワップを行えないのでこのような差が生じた．上限回数が 2 回，3 回の場合は Action-GDL とほぼ同等な UTIL メッセージサイズを得られている．

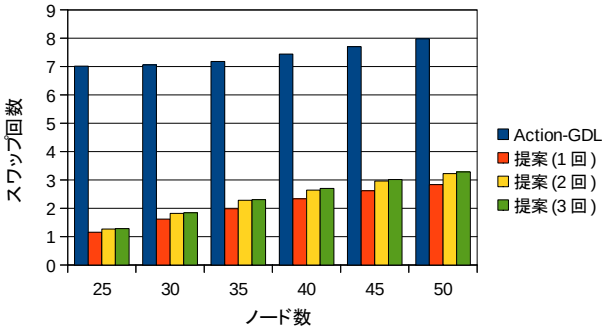

図 8: ノード数にたいするスワップ回数

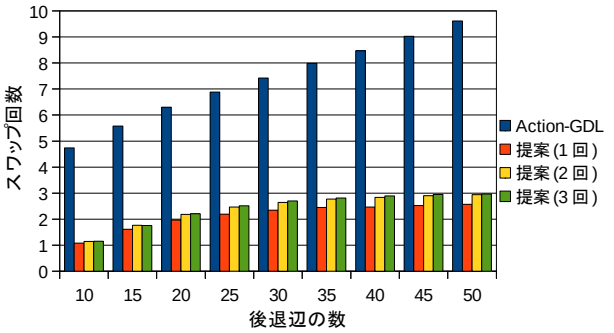

図 9: 後退辺にたいするスワップ回数

6.2. 情報漏洩

先ほどと同様にして，ノードの数を 40 個とし，後退辺の数を 10 本から 50 本まで 5 本ずつ増やしていった擬似木および，加える後退辺の数を 30 本とし，ノードの数を 25 個から 50 個まで 5 個ずつ増やしたいった擬似木に基づく JT に対し，Action-GDL と提案手法のスワップ回数を比較した．提案手法におけるスワップ上限回数を 1 回，2 回，3 回として評価した．結果を図 8 と図 9 に示す．

情報の漏洩はスワップを 1 回行う毎に 1 度起こる．結果では，後退辺の数が同じ場合，ノード数が多くなるにつれてスワップ回数が多く，提案手法のスワップ回数は既存手法の Action-GDL に比べて大幅に少ない．また，ノード数が多くなるにつれて，Action-GDL と提案手法のスワップ回数の差が縮まっている．これは，ノード数が増加するほど，メッセージサイズの削減が得られるスワップの箇所が多くなっているためであると考えられる．ノード数が同じである場合は，後退辺の数が多くなるにつれてスワップ回数が多くなった．また，提案手法のスワップ回数は既存手法の Action-GDL に比べて大幅に少なく，後退辺が多いほどスワップ回数の差は大きくなった．

提案手法を上限回数の別に見ると，上限回数がどの場

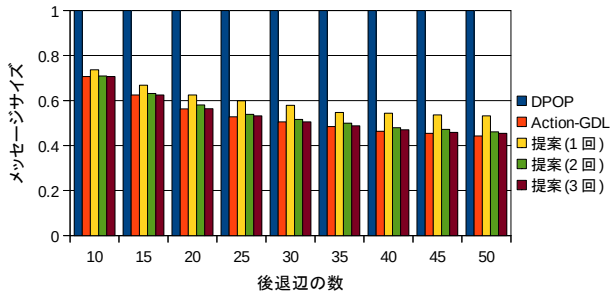

図 10: 後退辺にたいする UTIL メッセージサイズ

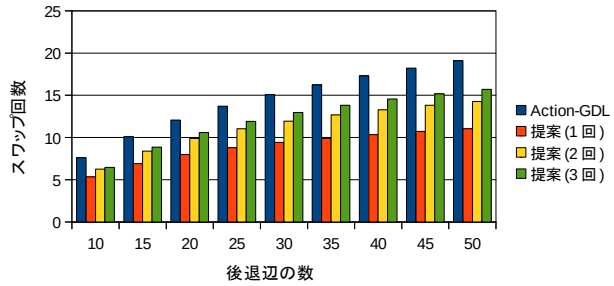

図 11: 後退辺にたいするスワップ回数

合でも既存手法の Action-GDL と大きな差がある．上限回数を 1 回としたときが最もスワップ回数を抑えられているが，上限回数が 2 回，3 回の場合との差は小さい．

6.3. 提案手法の効果

UTIL メッセージサイズとスワップ回数の結果から，提案手法で上限回数を 1 回としたとき，UTIL メッセージサイズは既存手法の Action-GDL に劣るが，その分スワップ回数は大きく減った．このことから，UTIL メッセージサイズをある程度維持しつつ情報の漏洩を抑制できたと考えられる．また，上限回数を 2 回，3 回としたときは，スワップ回数は上限回数を 1 回とした場合より多い．しかし，UTIL メッセージサイズは既存手法の Action-GDL とほぼ同等であり，かつ情報の漏洩も抑制できた．

UTIL メッセージサイズが既存手法の Action-GDL にそれほど劣らずにスワップ回数を抑制できる理由として，提案手法における，クリークの変数の個数を削減するスワップが大きく作用していると考えられる．すなわち，JT の変形において，メッセージサイズの削減が得られないスワップが多数あると考えられる．

6.4. スワップの上限回数

提案手法におけるスワップの上限回数を設けることで，スワップ回数を減らし，情報の漏洩を抑制できた．ここでは上限回数をどの程度に設定するかについて考える．先ほどの評価結果から，ノードの数，または擬似木の後退辺が多いほどスワップ回数が多くなる傾向がある．そして特に，擬似木の後退辺の数がスワップ回数に大きく関係していると考えられる．

ここでもう一点，木の深さに注目する．図 10 と図 11 は 40 個のノードを一列に繋げたものに対し，後退辺を任意の数加えた擬似木に基づく JT の UTIL メッセージサイズとスワップ回数を表す．ノードを一列に並べた理由は深さを得るためである．図 7 と図 10 から同じノードの数，加える後退辺の数が同じでも，木の構造によって評価結果が変わることが示された．UTIL メッセージサイズは図 7 より図 10 のほうが小さく，スワップ回数は図 9 より図 11 のほうが多い．また，木が深いほどスワップ回数が多くなり，その結果としてメッセージサイズが削減される程度も大きくなった．このことから，後退辺の数と木の深さから，スワップ上限回数を以下の表の指標によって設定できる可能性があると考えられる．

	木の深さ: 小	木の深さ: 大
後退辺の数: 小	上限回数: 小	上限回数: 中
後退辺の数: 大	上限回数: 中	上限回数: 大

この表は，後退辺が少なく，木が浅い場合はスワップ上限回数を少なく設定し，後退辺が多く，木が深い場合はスワップ上限回数を多く設定することを表している．後退辺の数と木の深さを指標とすることにより，既存手法の Action-GDL の UTIL メッセージサイズと同等のメッセージサイズを得られ，かつ十分に情報の漏洩を抑制できる適切なスワップ上限回数を設定できる可能性があると考えられる．

7. まとめ

本論文では，分散制約最適化問題 [3, 5, 6, 7, 10] の厳密解法における Action-GDL [10] に注目し，Junction-Tree [2] の変形の問題点である，情報の漏洩に対する提案手法として，クリークのスワップに新たな条件を追加した．提案手法および，既存手法の DPOP [8],Action-GDL を実験により比較し，提案手法の有効性を評価した．その結果，UTIL メッセージサイズは既存手法の Action-GDL にそれほど劣ることなく情報の漏洩を抑制することができた．また，変形ができない Junction-Tree を変形するために，辺を加える形式的な操作についても検討した．

今後の研究課題として，スワップ上限回数を設定するために後退辺の数と，木の深さの具体的な指標を決定することが挙げられる．また，集中処理化のための辺の追加については，従来手法のグラフ構造の変形における形式的な操作の拡張につながると考えられる．本研究では，適用例に基づく議論をしたが，このような辺の追加を行うアルゴリズム，および情報の漏洩のための変形の抑制手法との統合は，今後の研究課題として挙げられる．

謝辞

本研究の一部は，科研費 若手 (B)22700144 および平成 23 年度人工知能研究振興財団研究助成による．

参考文献

[1] Emma Bowring, Miling Tambe, and Makoto Yokoo. Multiply-constrained distributed con-

straint optimization. In *5th int. conf. on Autonomous agents and multiagent systems*, AAMAS '06, 2006.

[2] Finn V. Jensen and Frank Jensen. Optimal junction trees. In *10th int. conf. on Uncertainty in artificial intelligence*, pp. 360–366. Morgan Kaufmann Publishers Inc., 1994.

[3] Akshat Kumar, Boi Faltings, and Adrian Petcu. Distributed constraint optimization with structured resource constraints. In *8th int. conf. on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pp. 923–930. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[4] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *3rd int. conf. on Autonomous Agents and Multiagent Systems - Vloume 1*, AAMAS '04, pp. 310–317. IEEE Computer Society, 2004.

[5] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *3rd int. conf. on Autonomous Agents and Multiagent Systems - Vloume 1*, AAMAS '04, pp. 438–445. IEEE Computer Society, 2004.

[6] Pragnesh J. Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, Vol. 161, No. 1-2, pp. 149–180, January 2005.

[7] Adrian Petcu. A class of algorithms for distributed constraint optimization. In *2009 conf. on A Class of Algorithms for Distributed Constraint Optimization*, pp. 1–277. IOS Press, 2009.

[8] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *19th int. conf. on Artificial intelligence*, IJCAI'05, pp. 266–271. Morgan Kaufmann Publishers Inc., 2005.

[9] Atsushi Terauchi and Hideaki Suzuki. Multiagent system for efficiently propagating information through and across conjunctive communities. 情報処理学会論文誌, Vol. 40, No. 1, pp. 197–207, 1999.

[10] Meritxell Vinyals, Juan A. Rodriguez-Aguilar, and Jesús Cerquides. Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, Vol. 22, No. 3, pp. 439–464, 2011.

[11] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, Vol. 161, No. 1-2, pp. 55–87, January 2005.