分散制約最適化問題の効率的解法と動的問題への適用に関する研究

松井 俊浩

2006 年

目 次

第 1 章　序論 **9**

1.1　研究の背景 9

1.2　研究の目的 11

1.3　論文の構成 12

第 2 章　分散制約最適化問題 **15**

2.1　はじめに 15

2.2　分散制約充足問題 15

2.3　分散制約最適化問題 17

2.4　まとめ 19

第 3 章　分散制約最適化手法における上下界の導出と学習を用いた効率改善　21

3.1　はじめに 21

3.2　問題の形式化 22

3.3　従来手法 — 分枝限定法に基づく非同期分散探索手法 (ADOPT) . . 23

3.3.1　変数 25

3.3.2　コストの評価 26

3.3.3　変数に関する条件 27

3.3.4 メッセージ送信 29
3.3.5 メッセージ受信 30
3.3.6 最適解の検出およびアルゴリズムの終了 32
3.3.7 全体的な処理の構成 33
3.4 提案手法 1 — 上下界の導出と **COST** メッセージのショートカット 34
3.4.1 上下界についての部分解の分離 35
3.4.2 部分解の導出 36
3.4.3 **COST** メッセージの変更 37
3.5 提案手法 2 — 上下界の学習 39
3.5.1 LRU キャッシュの追加 39
3.5.2 部分解の内包関係に基づく統合 40
3.6 アルゴリズムの最適性について 41
3.7 評価 43
3.7.1 最大制約充足問題 44
3.7.2 重み付き制約充足問題 48
3.7.3 LRU キャッシュ長の影響 48
3.7.4 変数値配信パスの変更による影響 52
3.7.5 MPI 環境における評価 52
3.8 まとめ 55

第 4 章 深さ優先探索木に基づく分散制約最適化手法の動的問題への適用 57

4.1 はじめに 57
4.2 動的な分散制約最適化問題 59

4.3 動的な問題に追従する枠組み 60
4.4 制約の観測と制約網の構築 63
4.4.1 各ノードの状態の管理 63
4.4.2 関連ノードへの状態の通知 65
4.4.3 局所的な安定状態の検出 65
4.5 擬似的な木の構築 67
4.6 分散制約最適化手法の適用 68
4.6.1 論理時刻と解の番号による探索処理の識別 70
4.6.2 解の決定の同期 70
4.6.3 複数の問題の確保と前回結果の再利用 71
4.7 評価 72
4.7.1 解の決定と得られた解のコスト 76
4.7.2 解のコストの評価 77
4.7.3 制約網および木の構築 79
4.7.4 問題あたりのメッセージ数 79
4.8 まとめ 80
4.A ボトムアップな木の構築処理 83
4.B Adopt の処理の変更についての補足 85
4.B.1 終了処理の変更 85
4.B.2 実行開始の条件 85
4.B.3 メッセージ受信のタイミング 87
4.C 効率化手法の追加 88

第 5 章　擬似的な木の幅を考慮した動的計画法とバックトラックの統合　89

5.1　はじめに 89

5.2　分散制約最適化問題 91

5.3　従来手法 — 制約網に対する擬似的な木を用いた動的計画法 91

5.3.1　制約網に対する擬似的な木 91

5.3.2　擬似的な木に対する動的計画法の適用 93

5.4　提案手法 — 擬似的な木の幅を考慮した記憶のサイズの制限 96

5.4.1　擬似的な木の幅を考慮した記憶の制限とバックトラッキングの導入 96

5.4.2　バックトラックの範囲の削減 101

5.4.3　近似解法としての利用 102

5.5　評価 104

5.5.1　生成された擬似的な木と DPOP の評価 105

5.5.2　バックトラックによる遅延の評価 106

5.5.3　変数値を固定した場合の評価 107

5.6　関連研究と考察 107

5.7　まとめ 109

第 6 章　結論　111

謝辞　115

参考文献　117

本論文に関する研究業績等 **125**

第1章　序論

1.1　研究の背景

インターネット等のコンピュータ通信網の普及，およびコンピュータの高機能化・低価格化等により，通信網などを介した各種の分散システムが研究・開発されている．このような分散システムの一つの方式として，自律的な複数のノードが協調して処理を行うシステムは，特定のノードに一部の処理を集中する方式と異なり，対故障性や情報の分散化などの利点により応用が期待される．このようなシステムにおいては，目的とする処理をどのように分散協調処理としてモデル化するかが重要な課題であり，人工知能分野においては，分散協調問題解決, マルチエージェントシステムとして研究が行われている [1]．また，実際的な例として，複数のロボットの協調処理，センサ網における観測処理，携帯端末やネットワーク機器の資源割り当て処理などを目的とした様々な分散協調システムが研究されている．

一方で，個々の応用的な目的のための研究で得られた，分散アルゴリズム，プロトコルの設計は，他の目的への転用は十分に考慮されていないため，一般的な理解のための基礎的なモデルの必要性がある．このような基礎的なモデルとして，組み合わせ探索問題である制約充足問題 [2][3] を，分散環境における問題として適用し

た分散制約充足問題 [8] [4] [9] が提案された．特に分散環境における資源の割り当てなどの処理は組み合わせ探索問題として形式化できる．このため分散制約充足問題のような組み合わせ探索問題と分散探索アルゴリズムによる解法を用いた分散協調処理のモデルは重要であり，分散システムの基礎的な検討・理解のために活用できると考えられる．分散制約充足問題の基礎的な研究としては，N-Queens やランダムな 2 項制約網などを例題とし，解法の研究が行われてきた [9]．また，応用的な問題としてはセンサ網における観測資源の割り当て [48] などへの適用が提案されている．

しかし，より一般的な問題を扱う場合は，組み合わせ探索問題では無く，最適化問題としての形式化が必要であり，分散制約充足問題による形式化では記述に限界があるという課題があった．このため，分散制約充足問題の制約の一部を緩和する手法が導入された [11]．このような制約の緩和のコストを評価するために，各制約に重みを関連付ける手法や．制約に階層的な構造を導入する方法などが提案された [10][12]．近年，このような問題が，分散制約最適化問題として形式化され, その解法が提案されている [13][15][46].

特に，分散制約最適化問題の完全かつ健全な解法は，最適解を得ることが重要な問題に不可欠であることや，基礎的な解法の一つであることから，その検討は重要であると考えられる．一般に組み合わせ探索の基本的な解法として分枝限定法が用いられる．このような分枝限定法に基づく手法を分散アルゴリズムとして構成し，分散制約最適化問題の解法として適用する基本的な手法 [11] も示されたが，逐次型の探索を模倣する同期処理によるために，その探索の効率に課題があった．近年，このような分散分枝限定法を改良した，メモリ制限型の A*アルゴリズ

ムにもとづく分散制約最適化手法が提案された [15]．また，このような解法では，制約網に対する深さ優先探索木などの擬似的な木により，変数を順序付けする手法が導入されている．このような木構造は探索の効率化に有効であるだけでなく，分散アルゴリズムにおけるメッセージ配信や計算の基礎としても重要である．

このような解法の効率の改善を検討することは，分散制約最適化手法の基本的な理解や，応用を検討する上で重要であると考えられる．また，解法の応用のためには，単に探索手法を効率化するのみではなく，問題の観測から解の決定までの，探索の前後処理を含めた総合的な処理の枠組みを検討する必要がある．また，実際的な分散環境は動的な変化を伴うため，動的な問題への適用も必要である．

1.2 研究の目的

本研究では特に，近年提案された，制約網に対する深さ優先探索木をにもとづく分散制約最適化手法に [15] について注目し，その探索処理の効率の改善と，動的問題への適用について検討する．このような分散制約最適化手法は，1.1 でも述べたように，健全性と完全性を持ち最適解が得られる解法である点や，制約網に対する生成木を用いるという点で基礎的な分散制約最適化手法として重要であると考えられる．

上記の分散制約最適化手法の探索処理の効率について検討し，その探索処理の効率を改善するための手法を提案する．従来手法はメモリ制限のある A*アルゴリズムを基礎とするものであり，分枝限定法，動的計画法についての効率化手法を適用し，探索の効率を改善できる．本研究では，コストの下界と部分解の導出に

基づく Backjumping，および，部分解とコストの学習を導入することで，探索の効率が改善されること示す．

また，実際的な分散システムへとしての実装のための基礎的な検討として，動的な問題に追従するための基本的な枠組みを提案する．本研究では動的な問題として，各ノードが観測する制約が，ある期間の後に変化する場合を想定する．このような問題では，各ノードが観測する制約の変化を起点として，制約網の再構築，ノードの順序付け，探索処理の実行および解の決定の処理を反復的に実行する必要がある．これらをボトムアップな非同期分散処理として統合した枠組みを示す．

上記において検討する分散制約最適化手法 [15] を動的計画法に特化した解法として提案された手法 [46] についても検討する．このような解法はメッセージ数とメッセージサイクル数は擬似的な木の深さに対する多項式時間の複雑度となる．しかし，一方で空間複雑度は擬似的な木の幅に対して指数関数的に増大する．空間複雑度を削減するために擬似的な木の幅を考慮し，一部をバックトラック型の探索とするための基本的な手法を示す．

1.3 論文の構成

本論文の次章以降の構成は以下の通りである．

第 2 章では分散制約最適化問題についての概論を示す．まず，分散制約最適化問題の基礎となる，分散制約充足問題に関する研究について述べる．次に，より一般的な組み合わせ最適化問題として形式化された，分散制約最適化問題について述べる．分散制約充足問題から分散制約最適化問題への問題の拡張および，各問

題の解法についての関連研究を概観する．

第 3 章ではメモリ制限型の A*アルゴリズムに基づく非同期分散探索手法 [15] をもとに，探索処理の効率化手法について述べる．まず，対象とする問題の形式化および従来手法の概要を示し，提案手法として 2 つの効率改善手法を示す．１つめの効率改善手法として，バックトラッキングの効率化のために，コストの下界に対する部分解の導出と，逐次処理のバックトラック型の探索における Backjumping に類する手法を導入する．2 つめの効率改善手法として，複数の部分解とコストを学習する手法を導入する．提案手法の有効性について計算機実験により評価する．

第 4 章では第 3 章で検討した解法について，動的な環境への適用について述べる．まず，対象とする動的な問題を形式化し，提案手法を示す．本研究では特に，制約網の変化の観測，制約網の再構築，制約網に対する木の構築, 探索の実行と解の決定の一連の処理をボトムアップな非同期分散処理として統合する．このような一連の処理を反復的に動作させることで，動的に変化する制約網に対し，追従する枠組みを示す．例題を用い提案手法の有効性を評価する．

第 5 章では，関連研究に関する検討として，擬似的な木を用いる動的計画法に特化した解法として提案された分散制約最適化手法 [46] について，メモリ制限のため基本的な手法を検討する．深さ優先探索木などの擬似的な木による変数の順序に基づく探索手法に動的計画法を適用する場合，メッセージ数とメッセージサイクル数は木の深さに対する多項式時間の複雑度となるが，空間複雑度は擬似的な木の幅に対して指数関数的に増大する．この擬似的な木の幅を考慮し，木の幅が大きい箇所について，変数にバックトラックを導入する手法を示す．

第 6 章では本論文の結論として，以上の内容についてまとめ，今後の研究課題

について述べる.

第2章　分散制約最適化問題

2.1　はじめに

本章では分散制約最適化問題についての概論を示す．まず，分散制約最適化問題の基礎となる，分散制約充足問題に関する研究について述べる．次に，より一般的な組み合わせ最適化問題として形式化された，分散制約最適化問題について述べる．分散制約充足問題から分散制約最適化問題への問題の拡張および，各問題の解法についての関連研究を概観する．

2.2　分散制約充足問題

分散制約充足問題 (DCSP)[9] は，制約充足問題 (CSP))[2] を分散環境に適用したモデルであり．分散協調問題解決，マルチエージェントシステムの基礎的な理解・検討として研究されている．

基本的な DCSP は，次のように形式化される．システムはメッセージ通信を行う複数のノード (エージェント) から構成される．各ノード i は変数 x_i を持ち，変数 x_i は値域 D_i について $d \in D_i$ の値をとる．各変数 i は 2 項制約 $c_{i,j} \in C$ により，他の変数 x_j(ノード j の変数) と関係する．制約約 $c_{i,j}$ により値のペア $\{(x_i, d_i), (x_j, d_j)\}$

が評価される．全ての変数の値の割り当て $\mathcal{A}$ についての大域的な制約の評価は

$$\bigwedge_{\forall c_{i,j} \in C,\, \{(x_i,d_i),(x_j,d_j)\} \subset \mathcal{A}} c_{i,j}$$

で表される．目的とする解は全ての制約を充足する変数値の割り当て $\mathcal{A}(= \mathcal{A}^*)$ である．各ノードは自分の変数に関連する制約を知る．また，変数値の選択はその変数を持つノードのみが行う．各ノードは任意の他ノードとメッセージ通信を行う．メッセージ通信にもとづく分散アルゴリズムにより解を求める．

DCSP は CSP 同様に組み合わせ探索問題にもとづくモデルである．分散環境における資源の割り当てや制御など多くの処理は組み合わせ探索問題として形式化できるため，組み合わせ探索問題と分散探索アルゴリズムを用いた分散協調処理のモデルは重要であり，これらを分散システムの基礎的な検討・理解のために活用できると考えられる．

DCSP の基礎的な研究として，N-Queens やランダムな 2 項制約網などを分散環境に拡張した例題とし，解法の研究が行われてきた [9]．

完全性を持つ解法として，バックトラック型の探索にもとづく非同期バックトラッキングアルゴリズム [8]，CSP の解法である弱コミットメントアルゴリズム [22] を DCSP に適用した非同期弱コミットメントアルゴリズム [23]，などが提案された．

また，反復改善型のアルゴリズムとして，逐次型の Breakout 法 [24] を DCSP に適用した分散 Breakout 法 [25] などが提案された．

DCSP の応用的な問題として，センサ網における観測資源の割り当て [48] などへの適用が提案されている．

2.3 分散制約最適化問題

DCSP は組み合わせ探索問題に基づくモデルであり，その解法も組み合わせ探索問題を対象とする．

逐次型の CSP においては，より一般的な問題を扱う場合は，組み合わせ探索問題では無く，最適化問題としての形式化が必要であり，CSP に基づく形式化では記述に限界があるという課題があった．このため，過制約な CSP について一部の制約を緩和するように，拡張された問題が導入された [5]．また問題は制約の緩和におけるコストを評価するために制約に重みを付け，最適なコストを得る解を求める組み合わせ最適化問題として形式化された [6]．これらの問題についての基本的な解法として，分枝限定法に基づく解法などが適用された [5][6][7] ．

同様に，分散制約充足問題についても組み合わせ最適化問題へ拡張することが検討されてきた．このような拡張として，過制約な問題についてより多くの制約を充足する最大制約充足問題や，制約に階層的な構造を導入した制約充足問題がに基づく問題が提案され，これらの解法として分散制約充足問題の解法を反復的に実行する手法などが提案された [11][12]．また，分散制約充足に基づく解法を拡張する [13] 手法や，近似解法 [10] の適用が提案されている．

このような分散制約最適化問題 (DCOP) は DCSP よりも一般的な問題を形式化できることから，より応用的な問題を検討する上で重要であると考えられる．DCOP の解法は完全かつ健全な解法と，近似的な解法に分類されるが，特に，完全かつ健全な解法は，最適解を得ることが重要な問題に不可欠であることや，基礎的な解法の一つであることから，重要である．

基本的な DCOP は DCSP の制約にコスト評価のための関数を関連付けた問題と

して形式化される．システムはメッセージ通信を行う複数のノード (エージェント) から構成される．各ノード i は単一の変数 x_i を持ち，変数 x_i は値域 D_i について $d \in D_i$ の値をとる．各変数は 2 項制約で他の変数 x_j(ノード j の変数) と関係する．各制約に対応するコスト関数 $f_{i,j} : D_i \times D_j \rightarrow N$ により，値のペア $\{(x_i, d_i), (x_j, d_j)\}$ に対するコストが評価される．全ての変数についての値の割り当て $\mathcal{A}$ に対する大域的なコスト評価 $F(\mathcal{A})$ は，

$$F(\mathcal{A}) = \sum_{\forall\{(x_i,d_i),(x_j,d_j)\}\subset\mathcal{A}} f_{i,j}(d_i, d_j)$$

により表される．探索するべき最適解は $F(\mathcal{A})$ を最小化する割り当て $\mathcal{A}(= \mathcal{A}^*)$ である．

各ノードは自分の変数に関連する制約およびコスト関数を知る．また，変数値の選択はその変数を持つノードのみが行う．

逐次型の組み合わせ探索の基本的な解法として分枝限定法が用いられる．このような分枝限定法に基づく手法を分散アルゴリズムとして構成し，DCOP の解法として適用することが提案されたが，基本的に逐次型の探索を模倣する同期処理によるために，その探索の効率には課題があった．

これに対して，メモリ制限型の A*アルゴリズムにもとづく分散制約最適化手法が提案された [16]．また，この解法では，制約網に対する深さ優先探索木などの擬似的な木により，変数を順序付けする手法が導入されている．このような木構造は探索の効率化に有効であるだけでなく，分散アルゴリズムにおけるメッセージ配信や計算の基礎としても重要である．また，このような制約網に対する擬似的な木構造を動的計画法に特化した解法として提案された手法が提案された [46]．

DCOP の基礎的な例題としては重み付きのグラフ彩色問題などが用いられる [15]. また，応用的な問題としては，センサ網の資源割り当てやイベントスケジューリング問題が提案されている [17].

2.4 まとめ

本章では，まず，分散制約最適化問題の基礎となる，分散制約充足問題に関する研究について述べ，より一般的な組み合わせ最適化問題として形式化された，分散制約最適化問題について述べた．分散制約充足問題から分散制約最適化問題への問題の拡張および，各問題の解法について関連研究の概要を示した．

以下では，特に，擬似的な木にもとづく分散制約最適化手法および関連する手法を議論の対象とする．解法の探索の効率改善のための手法，および動的な環境への適用のための手法について述べる．

第3章　分散制約最適化手法における上下界の導出と学習を用いた効率改善

3.1　はじめに

本章では近年提案された分散制約最適化手法 [15][16] に関する効率改善手法について提案する．

この手法は，は分枝限定法/メモリ制限型の A*アルゴリズムにもとづく手法であり，制約網に対する深さ優先探索木により生成される変数順序を用いたコスト計算を基礎として，完全性，健全性をを持つ非同期並列探索を実現している．また，大域的なアルゴリズムの停止までの機構を持つこと，ユーザが指定したコストでの探索の打ち切りが可能であることなど，分散探索において基礎的な処理の幾つかが統合されている．また，問題の規模に対して多項式の空間複雑度である．一方で，手法の最悪の場合の計算複雑度は指数関数的であり，探索の効率化のために，適切なヒューリスティクスのを適用することが有効である．

上記の手法は特に比較的簡単なバックトラック処理にもとづくため，このバックトラック処理の冗長性を削減し探索を効率化する余地がある．また，記憶の制限

と非同期性により同一解を再探索する機会が多い．本研究ではバックトラックにおけるオーバヘッドの削減のために，上下界に関連する解の導出および上位ノードへの下界情報の通知のショートカットを導入する．また，再探索の削減のために，探索済みの上下界情報の学習を導入する．また，これらの効率改善手法により，従来手法の効率が改善することを計算機実験の結果により示す．

以下では先ず 3.2 で本章で対象とする分散制約最適化問題を示し，3.3 で従来手法の概要を示す．次に提案手法として，3.4 においてバックトラック効率改善のための上下界の導出と下界情報通知のショートカット，さらに 3.5 において探索済みの上下界情報の学習について述べる．3.6 で提案手法の最適性について述べ，3.7 において計算機実験による評価を示す．3.8 で本章の内容をまとめる．

3.2　問題の形式化

まず，本研究で対象とする分散制約最適化問題について簡単に形式化する．

システムはメッセージ通信を行う複数のノード (エージェント) から構成される．各ノード i は単一の変数 x_i を持ち，変数 x_i は値域 D_i について $d \in D_i$ の値をとる．各変数は 2 項制約で他の変数 x_j(ノード j の変数) と関係する．各制約に対応するコスト関数 $f_{i,j} : D_i \times D_j \rightarrow N$ により，値のペア $\{(x_i, d_i), (x_j, d_j)\}$ に対するコストが評価される．全ての変数についての値の割り当て $\mathcal{A}$ に対する大域的なコスト評価 $F(\mathcal{A})$ は，

$$F(\mathcal{A}) = \sum_{\forall\{(x_i,d_i),(x_j,d_j)\}\subset\mathcal{A}} f_{i,j}(d_i, d_j)$$

により表される．探索するべき最適解は $F(\mathcal{A})$ を最小化する割り当て $\mathcal{A}(= \mathcal{A}^*)$ で

ある．

各ノードは自分の変数に関連する制約およびコスト関数を知る．また，変数値の選択はその変数を持つノードのみが行う．以下では必要に応じて変数名をノード名と区別せずに表記する。

各ノードは任意の他ノードとメッセージ通信を行うものとする．通信においては同一ノード間のメッセージ順序は保存されるが，他ノード間のメッセージ順序は保存されない．ノードおよび通信における故障は考慮しない．

3.3 従来手法 — 分枝限定法に基づく非同期分散探索手法 (ADOPT)

提案手法の説明の準備として，従来手法である分枝限定法に基づく非同期分散探索手法 [15] について概要を示す．

この手法では，2 項制約に対応する制約網について，深さ優先探索木による変数 (ノード) の順序付けが，事前に生成されていることを前提とする．深さ優先探索木に基づく順序付けでは，各サブツリーに配置された変数の間に制約辺が存在しないため，サブツリー間の処理に並列性が得られる．木を生成する際のノードの展開順序には，最も近傍数が多いものを優先するヒューリスティクスを用いる．深さ優先探索木を生成するための基本的な分散アルゴリズムには [27], [28] [29] などがある．ノード i の親ノードを $parent_i$，子ノードの集合を $children_i$，また，木に含まれない制約辺も含めた上位/下位近傍の集合を $upperNeighbors_i/lowerNeighbors_i$ で表す．

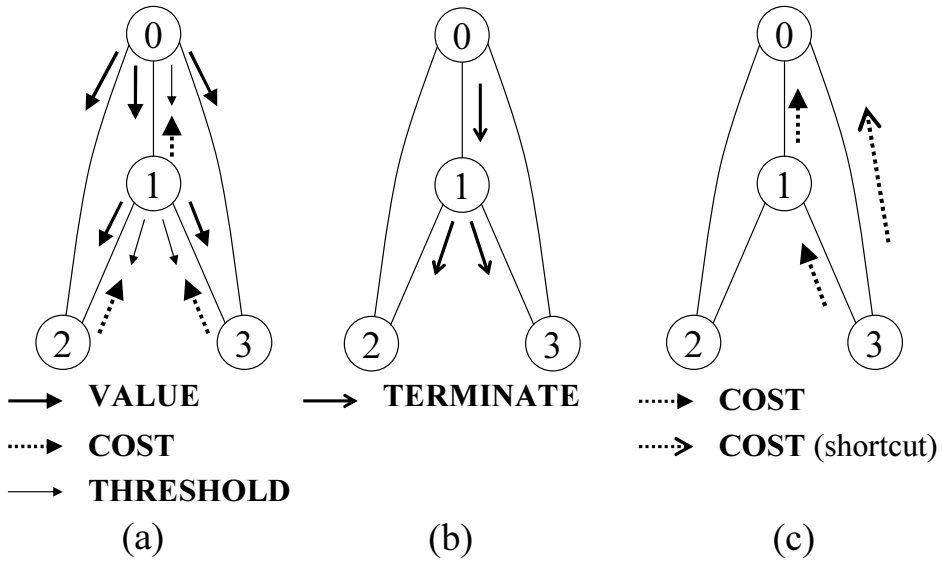


図 3.1: Adopt のメッセージ

この手法の処理の概要は次のとおりである．(1) 各ノードは解のコストと自ノードに分配されたコストに基づいて自変数の値を選択し，制約で関連する下位ノードに同報 (**VALUE** 送信) する．(2) 各ノードは解のコストを親ノードに通知 (**COST** 送信) する．(3) 各ノードは解のコストに基づいて，自ノードと子ノードでのコストの分配を決定し，子ノードに通知 (**THRESHOLD** 送信) する．以上の動作を反復し，最終的に根ノードではコストの上下界が一致する．このとき，根ノードは自変数の値を最適解に固定し子ノードに終了を通知 (**TERMINATE** 送信) する．子ノードは同様に最適解を発見し再帰的に全体が終了する．各メッセージの配信パスについて図 3.1(a)(b) に示す．

以下にアルゴリズムの構成要素を示す．本研究ではアルゴリズムの本質的な部分を整理するため，文献 [15] のルールおよび表現を修正した．

3.3.1 変数

各ノード i は自分の状態に関わる次の変数を持つ.

- d_i: 自ノードの変数値
- $threshold_i$: 変数値を変更するコストの閾値

初期値は $d_i \in D_i$, $threshold_i = 0$ である．d_i はコスト評価に応じて変更され，後述の **VALUE** メッセージにより，関連ノードに送信される．$threshold_i$ は，後述の **THRESHOLD** メッセージによりノード i 以下のサブツリーに割り当てられたコスト値であり，変数値を変更する基準となる.

また，ノード i は上位ノードの解のキャッシュとして

- $currentContext_i$: 上位ノードの解

を持つ．初期値は $currentContext_i = \{\}$ である．後述の **VALUE** および **COST** メッセージ受信処理により上位ノードの解が格納される.

下位ノードについては，コスト評価のキャッシュとして，$d \in D_i$, $x \in children_i$ について次の情報を持つ.

- $context_i(d, x)$: 子ノードの上位ノードの解
- $lb_i(d, x)$: 子ノードのコスト下界値
- $ub_i(d, x)$: 子ノードのコスト上界値
- $t_i(d, x)$: 子ノードの値変更閾値

それぞれの初期値は$context_i(d,x) = \{\}, lb_i(d,x) = 0, ub_i(d,x) = \infty, t_i(d,x) = 0$である．後述の**COST**メッセージ受信処理により子ノードのコスト評価が$context_i(d,x)$, $lb_i(d,x)$, $ub_i(d,x)$に格納される．また，$t_i(d,x)$はノードiがノードxに対して決定し，後述の**THRESHOLD**メッセージにより子ノードに送信される．

3.3.2　コストの評価

各ノードiが知る上位ノードの現在の部分解に関するコストの評価は以下のように定義される．

[**局所コスト** $\delta_i(d)$]

ノードiの，$d \in D_i$についての，上位近傍ノードに対する制約のコスト評価の総和を局所コスト$\delta_i(d)$とする．このように各ノードが上位近傍ノードの制約のみを評価することにより，評価の重複が避けられる．

$$\delta_i(d) = \sum_{\substack{(x_j,d_j)\in currentContext_i,\\ x_j \in upperNeighbors_i}} f_{i,j}(d,d_j)$$

[**各変数値についての上下界** $LB_i(d), UB_i(d)$]

ノードiを根とするサブツリーの$d \in D_i$についての上下界$LB_i(d), UB_i(d)$は，局所コストと各子ノードの上下界値の総和となる．

$$\begin{aligned} LB_i(d) &= \delta_i(d) + \sum_{x\in children_i} lb(d,x) \\ UB_i(d) &= \delta_i(d) + \sum_{x\in children_i} ub(d,x) \end{aligned}$$

[**上下界** LB_i, UB_i]

ノード i を根とするサブツリーの上下界 LB_i, UB_i は，$d \in D_i$ に対する上下界の最小値になる．

$$LB_i = min_{d \in D_i} LB_i(d)$$

$$UB_i = min_{d \in D_i} UB_i(d)$$

3.3.3 変数に関する条件

各ノード i は変数について以下の条件を維持しなければならない．

[$context_i(d, x)$ **条件**]

子ノードについての $context_i(d, x)$ と $currentContext_i$ が矛盾する場合, $context_i(d, x)$ に付随する $lb_i(d, x)$, $ub_i(d, x)$, $t_i(d, x)$ は無効である．従って $context_i(d, x)$ と $currentContext_i$ は矛盾してはならない．すなわち両者に含まれる同一の変数の値は異なってはならない．

$$\forall d \in D, x \in children,$$

$$context_i(d, x) \text{ と } currentContext_i \text{は矛盾しない}$$

もしも矛盾する場合は，$context_i(d, x)$, $lb_i(d, x)$, $ub_i(d, x)$, $t_i(d, x)$ を初期値にリセットする．

[$t_i(d, x)$ **条件 (ChildThresholdInvariant)**]

子ノードの閾値は子ノードの上下界を超えてはならない.

$$\forall d \in D, x \in children_i,$$
$$lb_i(d,x) \leq t_i(d,x) \leq ub_i(d,x)$$

もしも超える場合には $t_i(d,x)$ の値を上界または下界に制限する.

[$threshold_i$ **条件 (ThresholdInvariant)**]

自ノードの閾値は自ノードの上下界を超えてはならない.

$$LB_i \leq threshold_i \leq UB_i$$

もしも超える場合には $threshold_i$ の値を上界または下界に制限する.

[d_i **条件**]

変数値に対する下界の評価値は閾値を上回ってはならない．ただし，閾値と上界が等しい場合は，上界に対応する変数値でなければならない.

$$\begin{cases} UB_i(d_i) = UB_i & if \;\; threshold_i = UB_i \\ LB_i(d_i) \leq threshold_i & otherwize \end{cases}$$

下界の評価値が閾値より大きい場合，$LB_i(d_i) = LB_i$ となるよう，d_i を変更する．これは最良解優先探索を意図した値変更である．また，下界の評価値が閾値以下であれば値変更しないことにより，バックトラックが抑制される.

[$t_i(d,x)$ **条件 (AllocationInvariant)**]

自ノードの閾値は自ノードを根とするサブツリーに割り当てられたコストである．従って，自ノードの局所コストと子ノードの閾値の総和と等しくなければな

らない.

$$threshold_i = \delta_i(d_i) + \sum_{x \in children_i} t_i(d_i, x)$$

もしも等しくない場合には任意の $t_i(d, x)$ を増加または減少し，条件を維持する.

ただし，[$t_i(d, x)$ 条件 (ChildThresholdInvariant)] に矛盾してはならない.

3.3.4 メッセージ送信

各ノードは他ノードに以下のメッセージを送信する[1] .

[**VALUE 送信**]

送信内容: (**VALUE**,(x_i, d_i))

送信する条件: d_i が更新された

送信先: $x \in lowerNeighbors_i$

[**COST 送信**]

送信内容:

(**COST**,x_i,$currentContext_i$,LB_i,UB_i)

送信する条件:

($\forall x \in upperNeighbors_i$ より **VALUE** を受信した) $\wedge$

(d_i,$currentContext_i$, $threshold_i$, 子ノードキャッシュのいずれかが更新

された)

$\wedge$ (**TERMINATE** 未受信)

送信先: $parent_i$

[1] 文献 [15] にはメッセージの厳密な送信条件が明示されていない．本研究では文献とほぼ同程度の平均メッセージ数となるよう，アルゴリズムの動作に支障のない条件を設けた.

[**THRESHOLD 送信**]

送信内容:

(**THRESHOLD**,$currentContext_i$,$t_i(d_i, x)$)

送信する条件:

d_i,$currentContext_i$, 子ノードキャッシュのいずれかが更新された

送信先: $x \in children_i$

[**TERMINATE 送信**] [2]

送信内容:

(**TERMINATE**,

$currentContext_i \cup (x_i, d_i), t_i(d_i, x))$

送信する条件:

終了条件が成立した (後述)

送信先: $x \in children_i$

3.3.5　メッセージ受信

各ノードはメッセージ受信時に以下の処理を実行する.

[**VALUE 受信**]

(**VALUE**,(x_k, d_k)) 受信時:

if TERMINATE 未受信 **then**

[2]文献 [15] では，最後に親ノードで決定された $t_i(d_i, x)$ が子ノードの $currentContext_x$ と矛盾せずに受信されることが，アルゴリズムの最適性の条件の一部となっていると考えられる．本研究ではこの条件を明確にするため **TERMINATE** メッセージに $t_i(d_i, x)$ を加えた.

$currentContext_i$ に (x_k, d_k) を加える,

または置換する.

endif

[**COST 受信**]

(**COST**,x_k,$context_k$,lb_k,ub_k) 受信時:

$d \leftarrow (x_i, d) \in context_k$ である d

$context_k$ から (x_i, d) を除く.

if TERMINATE 未受信 **then** (2.5-A)

$\forall (x_j, d_j) \in context_k, x_j \notin upperNeighbors_i$

について,$currentContext_i$ に

(x_j, d_j) を加える, または置換する.

endif

[$context_i(d, x)$ 条件] を維持する. (2.5-B)

if $context_k$ と $currentContext_i$ が整合する

then [3] (2.5-C)

$\forall (x_j, d_j) \in context_k$ について,

$context_i(d, x_k)$ に (x_j, d_j) を加える,

または置換する.

[3]文献 [15] では，$context_k$ と $currentContext_i$ が整合すれば $context_i(d, x_k)$, $lb_i(d, x_k)$, $ub_i(d, x_k)$ にそれぞれ $context_k$,lb_k,ub_k を代入する記述となっている．しかし，複数のノードにおける子ノードキャッシュのリセットとメッセージの遅延により，$currentContext_i$ と整合する $context_k$,$lb_k < lb_i(d, x_k)$,$ub_i(d, x_k) < ub_k$ なる **COST** メッセージを受信し，コスト評価の単調性が維持されなくなる可能性があると考えられる．これはその後の摂動により解消されると考えられるが，本研究では，$currentContext_i$ と整合するかぎり，$context_i(d, x_k)$, $lb_i(d, x_k)$, $ub_i(d, x_k)$ の単調性が維持できるよう手続きを修正した．以上の修正による探索効率への影響は大きくないことを予備実験により確認した.

if $lb_k > lb_i(d, x_k)$ **then**

$lb_i(d, x_k) \leftarrow lb_k$ **endif**

if $ub_k < ub_i(d, x_k)$ **then**

$ub_i(d, x_k) \leftarrow ub_k$ **endif**

endif

(上記 (2.5-B) の [$context_i(d, x)$ 条件] の維持により，$context_i(d, x)$ が $currentContext_i$ と矛盾する場合は，$context_i(d, x)$, $lb_i(d, x)$, $ub_i(d, x)$, $t_i(d, x)$ を初期値にリセットする)

[**THRESHOLD 受信**]

(**THRESHOLD**,$context_k$,t_k) 受信時:

if $context_k$ と $currentContext_i$ が整合する

then $threshold_i \leftarrow t_k$ **endif**

[**TERMINATE 受信**]

(**TERMINATE**,$context_k$,t_k) 受信時:

$currentContext_i \leftarrow context_k$

$threshold_i \leftarrow t_k$

TERMINATE 受信を記録する

3.3.6　最適解の検出およびアルゴリズムの終了

根ノード i ではいずれ $LB_i = threshold_i = UB_i$ となる．このとき，根ノードは変数値を最適解に固定し，終了することができる．ノード j の上位ノードが変数

値を固定して終了した後で，$threshold_j = UB_j$ となれば，ノード j は変数値を最適解に固定し，終了することができる．したがって最適解の検出および終了の判定は次のようになる．

[**終了判定**]

if $threshold_i = UB_i \wedge$

(i が根ノード$\vee$**TERMINATE** を受信済)

then

([d_i 条件] により $UB_i(d_i) = UB_i$)

終了条件成立を記録する

endif

3.3.7 全体的な処理の構成

以上の各条件の維持およびメッセージ送受信の処理を，手続き的に構成する場合の順序は一意では無い．本研究では各ノードは，メッセージ受信，各条件の維持/判定，メッセージ送信を反復するサイクル動作とした．

[初期化]

while 終了条件が成立していない **do**

(メッセージ受信)

while 受信メッセージ有り $\wedge$

終了条件が成立していない **do**

[**VALUE** 受信] **or** [**COST** 受信] **or**

[**THRESHOLD** 受信] **or**

[**TERMINATE** 受信]

enddo

(各条件の維持/判定を以下の順で実行)

[$context_i(d, x)$ 条件]

[$t_i(d, x)$ 条件 (ChildThresholdInvariant)]

[$threshold_i$ 条件 (ThresholdInvariant)]

[d_i 条件]

[$t_i(d, x)$ 条件 (AllocationInvariant)]

[終了判定]

(メッセージ送信)

[**VALUE** 送信]

[**COST** 送信]

[**THRESHOLD** 送信]

[**TERMINATE** 送信]

enddo

3.4　提案手法 1 — 上下界の導出と COST メッセージのショートカット

上記のアルゴリズムは深さ優先探索木による変数順序を用いたコスト計算を基礎としている．これにより，非同期分散探索処理の最適解への収束が保証される．

しかし，この手法では単純なバックトラック動作による処理の冗長性が問題となる．本節では，バックトラックにおけるオーバヘッドの削減を目的とした，上下界の導出と **COST** メッセージのショートカットによる効率化手法を提案する．ここで，提案手法では，(1) 各ノードは根ノードまでの経路に関する知識を持つこと，(2) 制約辺に沿わない任意の通信経路が存在することを前提とする．また，議論を簡単にするため，(3) 終了ノードへのメッセージは無視されるものとする．

3.4.1 上下界についての部分解の分離

従来手法では，子ノードについての上位ノードの解のキャッシュとして $context_i(d,x)$ を保持している．この情報は $lb_i(d,x)$,$ub_i(d,x)$ の根拠となる部分解である．しかし，下界の証明に必要な部分解は，上位ノード全ての変数割り当てを含まない場合もある．そこで，$context_i(d,x)$ を，$context_i^{lb}(d,x)$ および $context_i^{ub}(d,x)$ に分離し，$lb_i(d,x)$,$ub_i(d,x)$ の証明に必要な最低限の部分解を導出する．これは DCSP における違反解の導出 [19][20] に類似する操作であると考えられる．もしも，下界に関する部分解に親ノードが含まれない場合，深さ優先探索木をショートカットしたコスト通知 (図 3.1(c)) を行うことで，バックトラックにおける冗長性を改善できる．これは制約最適化問題における Backjumping[5] に類似する操作であると考えられる．

3.4.2　部分解の導出

変数集合 $X'_d \subseteq upperNeighbors_i$, $X''_d \subseteq children_i$ について,

$$\forall d \in D_i,$$

$$LB_i \leq \sum_{\substack{x_j \in X'_d, (x_j, d_j) \in \\ currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in X''_d} lb_i(d, x)$$

であるとき下界に関わる部分解 $context_i^{lb}$ は,

$$\bigcup_{d \in D_i} \quad (\{(x_j, d_j) \in currentContext_i | x_j \in X'_d\}$$

$$\cup \bigcup_{x \in X''_d} context_i^{lb}(x, d))$$

である.

このような部分解の導出の基準としては，(1) 小さいサイズの部分解であること，(2) より上位のノードの変数値のみを含むことなどが有効であると考えられる．本研究では簡単な導出手法として次の方法を用いた.

各変数値 $d \in D_i$ について，まず局所コストについて評価し，$f_{i,j}(d, d_j) > 0$ なる上位近傍 j の変数 x_j を X'_d に加え，$f_{i,j}(d, d_j)$ をコストとして合計する．このときコストが LB_i に達した時点で導出を終了する．全ての上位近傍についてのコストの合計が LB_i に満たない場合は，各子ノード $x \in children_i$ について $lb_i(d, x) > 0$ なる x を X''_d に加え，$lb_i(d, x)$ をコストとして合計する．このときコストが LB_i に達した時点で導出を終了する.

一方，上界に関わる部分解 $context_i^{ub}$ はノード i 以下のサブツリーに関わる全て

の制約に依存するため，

$$\{(x_j, d_j) | (x_j, d_j) \in currentContext_i, x_j \in upperNeighbors_i\} \cup$$
$$\bigcup_{d \in D_i, x \in children_i} context_i^{ub}(x, d)$$

である．従って，メッセージの遅延と，子ノードのキャッシュがリセットされることを無視すれば，ほぼ $currentContext_i$ と等しい．

3.4.3 COST メッセージの変更

上下解について分離された部分解を送信するため，**COST** メッセージの形式および送信先を変更する．親ノードへの **COST** メッセージは上下界に関する部分解を含み，次のようになる[4]．

(**COST**,x_i,$currentContext_i$, $context_i^{lb}$,$context_i^{ub}$,LB_i,UB_i)

もしも，$context_i^{lb}$ が親ノードを含まない場合，親ノードへの **COST** メッセージに加えて，下界に関する **COST** メッセージをショートカットして送信する．ただし，$LB_i = 0$ の場合は明らかに冗長なので送信しない．送信先 j は $context_i^{lb}$ に含まれる最下位の変数 x_j を持つノードとする．また，メッセージ中の変数 x_k は j の子ノードで i の祖先ノードのものとする．UB_i については上界および原因となる部分解を証明できないため，デフォルトの上界値 ∞ および部分解 {} とする．送信される **COST** メッセージは次のようになる．

(**COST**,x_k,$context_i^{lb}$, $context_i^{lb}$,{},LB_i,∞)

[4] ここでは簡単のために従来の $currentContext$ はそのままとし，$context_i^{lb}$,$context_i^{ub}$ を追加した

COST メッセージ受信処理の上下界の更新 (2.5-C) において，部分解を別に記録するよう変更する．下界の更新については以下のようになる．上界についても同様に $ub_i(x,d)$ が単調に減少するように更新する．

if $context_k$ と $currentContext_i$ が整合する
then
if $context^{lb}_k$ が (x_i,d') を含む **then**
$d' \leftarrow (x_i,d') \in context^{lb}_k$ である d'
$context^{lb}_k$ から (x_i,d') を除く．
endif
$context^{lb}_k$ が (x_i,d') を含んでいた場合
$d=d'$ について，
そうでなければ $\forall d \in D_i$ について
if $lb_k > lb_i(d,x_k)$ **or** $(lb_k = lb_i(d,x_k)$
and $|context^{lb}_k| < |context^{lb}_i(d,x_k)|)$
then
$lb_i(d,x_k) \leftarrow lb_k$
$context^{lb}_i(d,x_k) \leftarrow context^{lb}_k$
endif
endif

また，$[context_i(d,x)$条件$]$ も上下界に分離し，それぞれ$context^{lb}_i(d,x)$ と $lb_i(d,x)$，および $context^{ub}_i(d,x)$ と $ub_i(d,x)$ について別に $currentContext_i$ との整合性を維持する．$t_i(d,x)$ は $lb_i(d,x)$ により押し上げられる形で増加するため，$context^{lb}_i(d,x)$

が不整合の場合は $lb_i(d,x)$ とともにリセットする．

3.5 提案手法 2 — 上下界の学習

従来手法は多項式の記憶複雑度であり，各ノード i は，各変数値 d に対する子ノード x の上位ノードの解を $context_i^{lb}(d,x)$ または $context_i^{ub}(d,x)$ (以下 $context_i^{lb/ub}(d,x)$ と表記) についてのみ記録する[5]．これらが現在の解と矛盾する場合には初期値にリセットする．しかし，探索の間にリセットが頻発することは，冗長な探索が発生し探索サイクル数が増加する原因となる．十分な記憶が利用可能であれば，これまでに得られた部分解のコストを記録し再利用することで冗長な探索を削減することが適当である．そこで，本節ではコスト情報のリセットの回数を削減するための上下界の学習を導入する．このような手法では，CSP/DCSP における違反解の記録 [21][22][23] と同様に，理想的には指数オーダの記憶が必要となる点が問題となる．しかし，従来手法の完全性により，許容できるサイズの記憶で妥協し，補助的な効率化手法として導入することができる．

3.5.1 LRU キャッシュの追加

各ノード i の，$context_i^{lb/ub}(d,x)$ を複数記録することでリセットの回数を削減できる．しかし，これまでに探索した部分解を全て記録することは不可能であるため，LRU により複数の部分解を管理する．$context_i^{lb/ub}(d,x)$ ごとに有限長のバッファを設け，先頭要素を $context_i^{lb/ub}(d,x)$ として用いる．$currentContext_i$ と先頭

[5] ここでは従来手法に前述の提案手法 1 を適用したものをさらに拡張する形式で表記する

要素が矛盾する場合には，バッファ内に矛盾しない要素が存在すればそれを先頭に移動し，存在しなければ新規要素を先頭に挿入する．バッファの制限長を超えた要素は削除される．

3.5.2　部分解の内包関係に基づく統合

LRU による記憶を追加した場合であっても，キャッシュ内の整合性のある上下界情報 $context_i^{lb/ub}(d,x)$, $lb/ub_i(d,x)$ を上書きする形で変更する方法では，これまでに得られた上下界の情報が失われる機会が多い．整合性のある複数の上下界情報により，より広い探索空間の上下界を記録することが望ましいと考えられる．そこで，解と上下界の内包関係にもとづいて解を統合する．すなわち，上下界 $context^{lb/ub}$, lb/ub および $context'^{lb/ub}$, lb'/ub' について，

$$lb \geq lb' \wedge context^{lb} \subseteq context'^{lb}$$

$$ub \leq ub' \wedge context^{ub} \subseteq context'^{ub}$$

であれば，$context^{lb/ub}, lb/ub$ は $context'^{lb/ub}, lb'/ub'$ を置換する．そうでなければ置換されず，両者とも記録される．

COST メッセージ受信において上記の条件によるキャッシュ要素の置換または追加を行う．また，コスト評価で用いるキャッシュ内の先頭要素は，$currentContext_i$ と整合し上下界を最も狭くするものを選択する．

3.6 アルゴリズムの最適性について

提案手法では従来手法に対し，上下界に対応した部分解の導出，下界の通知のショートカット，複数の上下界の記録および統合を追加した．これらは従来手法の最適性を損なわない．

部分解の導出は，上界については従来手法と同様である．下界の部分解 $context_i^{lb} \subseteq currentContext_i$ の導出については，$X'_d \subseteq upperNeighbors_i, X''_d \subseteq children_i$ から，

$$
\begin{aligned}
&\forall d \in D_i, \\
&\sum_{\substack{x_j \in X'_d, xj \in \\ currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in X''_d} lb_i(d, x) \leq \\
&\sum_{\substack{x_j \in upperNeighbors_i, \\ xj \in currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in Children} lb_i(d, x) = \\
&\delta(d) + \sum_{x \in Children} lb_i(d, x) = LB_i(d)
\end{aligned}
$$

であり．下界の定義 $LB_i = \min_{d \in D_i} LB_i(d)$ および導出の条件，

$$
\forall d \in D_i, LB_i \leq \sum_{\substack{x_j \in X'_d, xj \in \\ currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in X''_d} lb_i(d, x)
$$

から，$context_i^{lb}$ のみについての下界のコスト評価は，従来手法による $currentContext_i$ についての下界 LB_i に等しい．

下界のショートカットについては次のとおりである．ノード l から祖先ノード u に，$context_l^{lb}$ がショートカットして通知されたとする．ノード u と l の間のノードの持つ変数の集合を $X_{u,l}$ で表す．各ノードが評価する評価関数の集合 F_* のうち，

$X_{u,l}$ に関するものからなる部分集合を $F_{u,l}$，その補集合を $\overline{F}_{u,l}$ で表す．ノード i の変数値 d，評価関数の集合 F，部分解 $context$ によりコストの一部が評価されることを $(d_i, F, context)$ で表す．各評価関数は 1 度のみ評価され加算されるため，ノード i の $LB_i(d)$ の定義は，次のように，$F_{u,l}$ および $\overline{F}_{u,l}$ についての評価に分けることができる．

$$\begin{aligned}
\delta_i(d_i) = \ & \delta_i(d_i, F_{u,l}, context) + \\
& \delta_i(d_i, \overline{F}_{u,l}, context)
\end{aligned}$$

$$\begin{aligned}
lb_i(d_i, x) = \ & lb_i(d_i, x, F_{u,l}, context) + \\
& lb_i(d_i, x, \overline{F}_{u,l}, context)
\end{aligned}$$

$$\begin{aligned}
LB_i(d_i) = \ & LB_i(d_i, F_{u,l}, context) + \\
& LB_i(d_i, \overline{F}_{u,l}, context)
\end{aligned}$$

$$LB_i(d_i, F_{u,l}, context) = \delta_i(d_i, F_{u,l}, context) + \sum_{x \in children_i} lb_i(d_i, x, F_{u,l}, context)$$

$$LB_i(d_i, \overline{F}_{u,l}, context) = \delta_i(d_i, \overline{F}_{u,l}, context) + \sum_{x \in children_i} lb_i(d_i, x, \overline{F}_{u,l}, context)$$

$context_l^{lb}$ は $X_{u,l}$ についての部分解を含まないから，$context_l^{lb} \subseteq currentContext_l$ についての $LB_l(d_l, F_*, context_l^{lb})$ は，$LB_l(d_l, F_*, context_l^{lb}) = LB_l(d_l, \overline{F}_{u,l}, context_l^{lb}) \leq LB_l(d_l, \overline{F}_{u,l}, current\text{-}Context_l)$ でなければならない．l の祖先で u の子ノードを k とすると，$currentContext_l \subseteq currentContext_k \cup \bigcup_{x_j \in X_{u,l} - \{x_k\}} \{(x_j, d_j)\}$, $d_j \in D_j$ のとき，$LB_l(d_l, \overline{F}_{u,l}, currentContext_l) \leq LB_k(d_k, \overline{F}_{u,l}, currentContext_k)$ である．

任意の $X_{u,l}$ に関する部分解について $LB_k(d_k, \overline{F}_{u,l}, currentContext_k) \leq LB_k(d_k, F_*, currentContext_k)$ であるから，ショートカットして通知された下界 $LB_l(d_l, F_*, context_l^{lb})$ は k 以下のサブツリーの下界を超えない．

複数の解の記録および統合では，これまでに得られた最小 (大) の上 (下) 界を保存するのみであるから，従来手法により最終的に得られる上下界を超えず，また，上 (下) 界は単調に減少 (増加) する．

以上より，提案手法により得られる上下界は，従来手法により最終的に得られる上下界を超えないため，従来手法の最適性を失わない．また，上 (下) 界は単調に減少 (増加) するよう統合されるため，提案手法により得られた上下界が最終的な上下界に満たない場合は，従来手法の条件によって得られた上下界により，結局は最終的な上下界に収束する．

3.7 評価

提案手法の有効性を計算機実験により評価した．評価用の問題は従来手法と同様に 3 色のグラフ彩色問題を用いた．各ノードは 3 値変数 1 個を持ち，他ノードと 2 項制約で関係する．制約数はノード数 n に対して，パラメータ $d = \{2, 3\}$ により $n \cdot d$ とした．$d = 2$ の場合は充足可能な問題が比較的多く，$d = 3$ の場合は過制約な問題が多くなる．従来手法 (normal) および，上下界の導出と **COST** メッセージのショートカット (shortcut)，さらに LRU キャッシュを追加したもの (sc+lru)，さらに内包関係に基づく解統合を追加したもの (sc+lru+integrate) について比較した．各制約の重みを全て 1 とした問題 (最大制約充足問題) および $\{1, \cdots, 10\}$ とし

た問題について実験を行った．以上の各条件について 25 問の問題を生成し結果を平均した．また，LRU キャッシュのサイズは各 (d, x) について 100 を上限とした．

実験では，まず，シミュレーションにより分散システムを模倣した．システム全体はメッセージ交換と各ノードの処理から成るサイクルで動作し，各ノードは各サイクルごとに，メッセージ受信，条件維持/判定，メッセージ送信の処理を反復するものとした．また，シミュレーションで用いたプログラムを MPI 上に移植し，非同期分散環境下での挙動を確認した．

3.7.1　最大制約充足問題

各制約の重みを 1 とした場合のサイクル数を図 3.2,3.3 に示す．d=2 の場合，メッセージのショートカットの効果と比較して，さらに上下界の学習を追加した場合の効果が小さい．これは，再探索の機会が少ないためであると考えられる．一方，d=3 の場合，上下界の学習による再探索削減の効果が得られた．

ただし，表 3.3 に示すように制約辺の密度 d によるランダムな問題の生成方法では，問題の最適コストのばらつきのため，サイクル数の平均に対する分散は大きい．また，同様の理由により図 3.2, および後述の図 3.5,3.8(d=2 の場合) のグラフでは，比較的最適コストの大きい問題が多く生成されたノード数 12,14 の問題で，バックトラックと再探索の影響でサイクル数の平均が増加している．改善の程度の詳細として，各問題についての，各提案手法の追加前後でのサイクル数の比較を表 3.1,3.2 に示す．各 25 問についてサイクル数が減少した場合 (<)，等しかった場合 (=)，増加した場合 (>) について，各場合の問題数および，各場合についてのサイクル数の最大，最小値を分類した．各提案手法は多くの問題において追加前

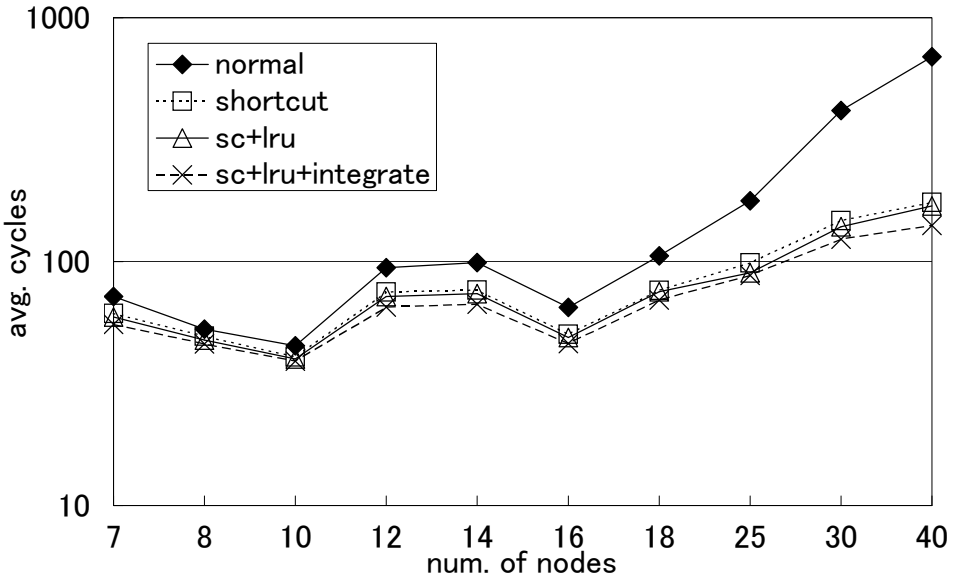


図 3.2: サイクル数 (制約重み=1,d=2)

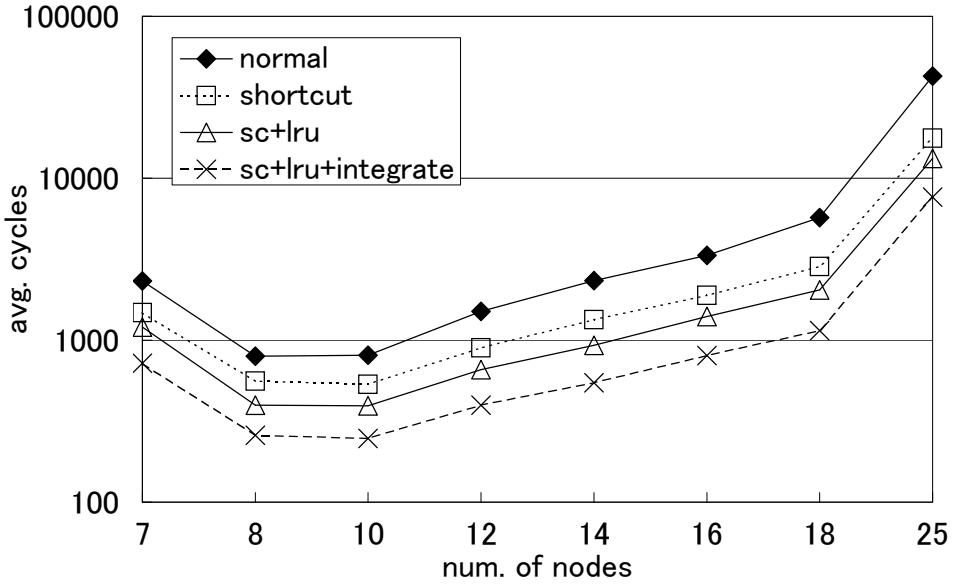


図 3.3: サイクル数 (制約重み=1,d=3)

表 3.1: 各問題のサイクル数の比較 (制約重み=1,d=2, 括弧内の手法は比較対象)

nodes	results			cycles					
	<	=	>	<		=		>	
				max	min	max	min	max	min
normal (. normal)									
7	0	25	0			175	16		
8	0	25	0			146	13		
10	0	25	0			93	14		
12	0	25	0			323	17		
14	0	25	0			275	17		
16	0	25	0			236	19		
18	0	25	0			357	27		
25	0	25	0			1249	28		
30	0	25	0			2120	36		
40	0	25	0			5207	48		
shortcut (. normal)									
7	17	7	1	138	35	66	16	54	54
8	8	17	0	111	42	89	13		
10	9	15	1	75	42	89	14	79	79
12	15	9	1	253	22	80	17	98	98
14	14	11	0	211	24	113	17		
16	11	13	1	169	30	51	19	34	34
18	17	8	0	239	27	33	27		
25	19	6	0	611	35	60	28		
30	21	4	0	839	37	88	36		
40	22	1	2	871	46	48	48	79	63
sc+lru (. shortcut)									
7	11	14	0	130	38	65	16		
8	7	17	1	100	34	74	13	69	69
10	6	19	0	73	41	89	14		
12	8	16	1	227	69	105	17	115	115
14	7	16	2	171	79	143	17	132	127
16	6	19	0	156	30	117	19		
18	8	14	3	213	28	96	27	161	138
25	8	17	0	508	54	66	28		
30	11	11	3	706	47	101	36	274	60
40	4	15	6	818	301	118	48	468	48
sc+lru+integrate (. sc+lru)									
7	15	8	2	112	35	56	16	41	36
8	8	16	1	86	32	74	13	39	39
10	5	18	2	71	42	56	14	92	71
12	13	9	3	174	21	51	17	127	66
14	13	10	2	148	26	82	17	146	82
16	16	9	0	134	23	70	19		
18	12	10	3	168	30	53	27	126	30
25	9	9	7	250	28	63	30	541	30
30	14	6	5	580	46	67	36	220	40
40	16	5	4	660	52	63	48	106	57

表 3.2: 各問題のサイクル数の比較 (制約重み=1,d=3, 括弧内の手法は比較対象)

nodes	results			cycles					
	<	=	>	<		=		>	
				max	min	max	min	max	min
normal (. normal)									
7	0	25	0			2316	2316		
8	0	25	0			1332	445		
10	0	25	0			1996	81		
12	0	25	0			3636	210		
14	0	25	0			7143	417		
16	0	25	0			11199	191		
18	0	25	0			19769	272		
25	0	25	0			111535	6130		
shortcut (. normal)									
7	25	0	0	1481	1481				
8	25	0	0	967	298				
10	25	0	0	1132	45				
12	25	0	0	2116	160				
14	25	0	0	3346	286				
16	25	0	0	8128	100				
18	25	0	0	9611	130				
25	25	0	0	62820	2818				
sc+lru (. shortcut)									
7	25	0	0	1207	1207				
8	23	0	2	725	260			351	328
10	23	1	1	835	142	45	45	182	182
12	24	0	1	1418	123			661	661
14	25	0	0	2396	200				
16	25	0	0	5662	96				
18	23	0	2	7991	317			1548	131
25	25	0	0	41010	2013				
sc+lru+integrate (. sc+lru)									
7	25	0	0	719	719				
8	25	0	0	433	197				
10	24	1	0	467	107	45	45		
12	25	0	0	933	111				
14	25	0	0	1196	177				
16	25	0	0	3064	91				
18	24	0	1	4771	121			482	482
25	25	0	0	25319	1162				

の手法と比較して，サイクル数が同等か改善した結果が得られた．また，比較的サイクル数を要する問題については，多くの問題でサイクル数の改善が得られた．これらは，後述の他の実験のサイクル数についても同様である．

d=3 の場合についてメッセージ数を図 3.4 に示す．提案手法では **COST** メッセージのショートカットの影響で，メッセージ数が増加している．しかし全体としての増加は比較的小さい．

3.7.2　重み付き制約充足問題

各制約の重みを $\{1, \cdots, 10\}$ とした場合のサイクル数を図 3.5,3.6 に示す．各制約の重み 1 の場合と同様に，d=2 の場合はメッセージのショートカットの効果が得られ，d=3 の場合はメッセージのショートカットおよび上下界の学習の効果が得られた．

3.7.3　LRU キャッシュ長の影響

各制約の重み 1,d=3, 内包関係に基づく解統合の場合について，キャッシュ長を $\{10, 20, 40, 80, 100\}$ とした場合のサイクル数への影響を図 3.7 に示す．サイズが増加するに従い，サイクル数が減少している．しかし，解の数は下位ノードでは指数関数的に増加するため，線形にキャッシュのサイズを増加しても得られる効果は次第に小さくなる．

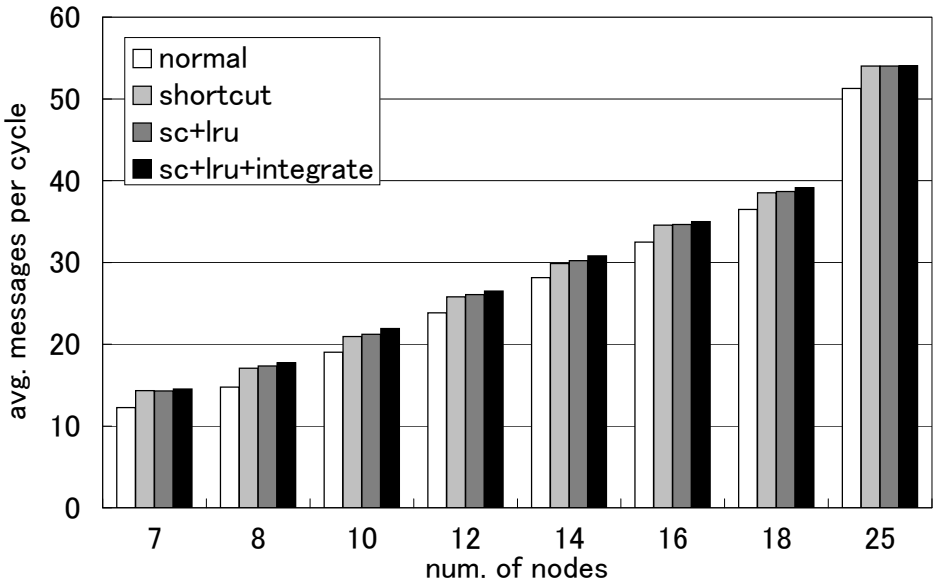


図 3.4: サイクルあたりのメッセージ数 (制約重み=1,d=3)

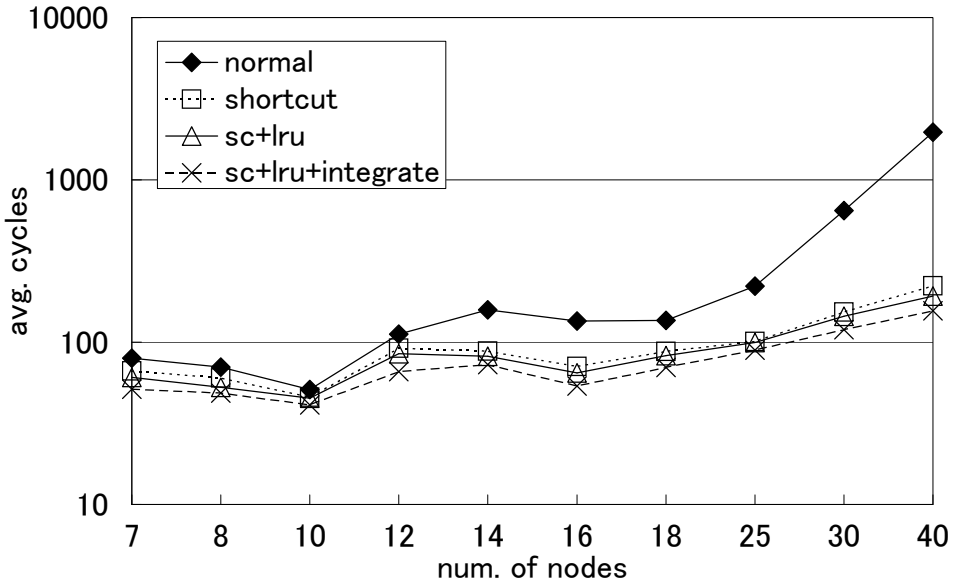


図 3.5: サイクル数 (制約重み=$\{1, \cdots, 10\}$,d=2)

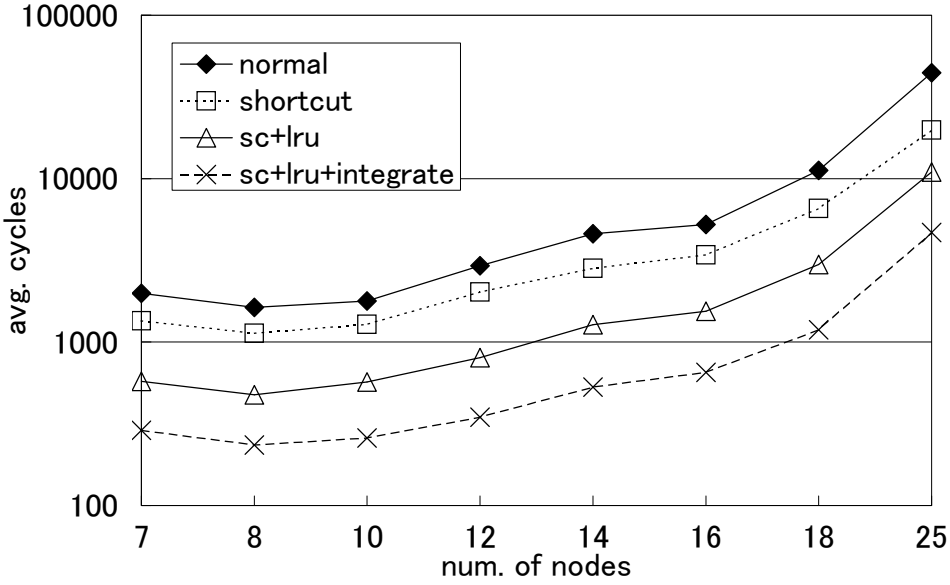


図 3.6: サイクル数 (制約重み=$\{1, \cdots, 10\}$,d=3)

表 3.3: 各結果の分散 (ノード数 16)

weight of constraint	1		1...10	
d	2	3	2	3
cycle				
normal	3606	8645435	55839	22302935
shortcut	1372	3453270	7976	10884349
sc+lru	1210	1757614	5001	1856625
sc+lru+int.	970	560947	2114	260599
messages per cycle				
normal		13		
shortcut		17		
sc+lru		20		
sc+lru+int.		25		

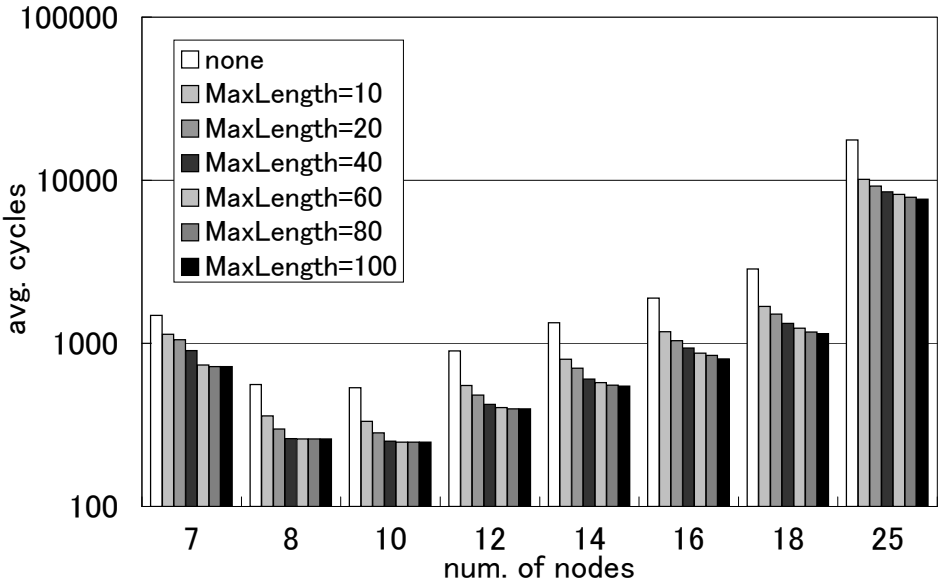


図 3.7: キャッシュ長のサイクル数への影響 (sc+lru+integrate, 制約重み=1,d=3)

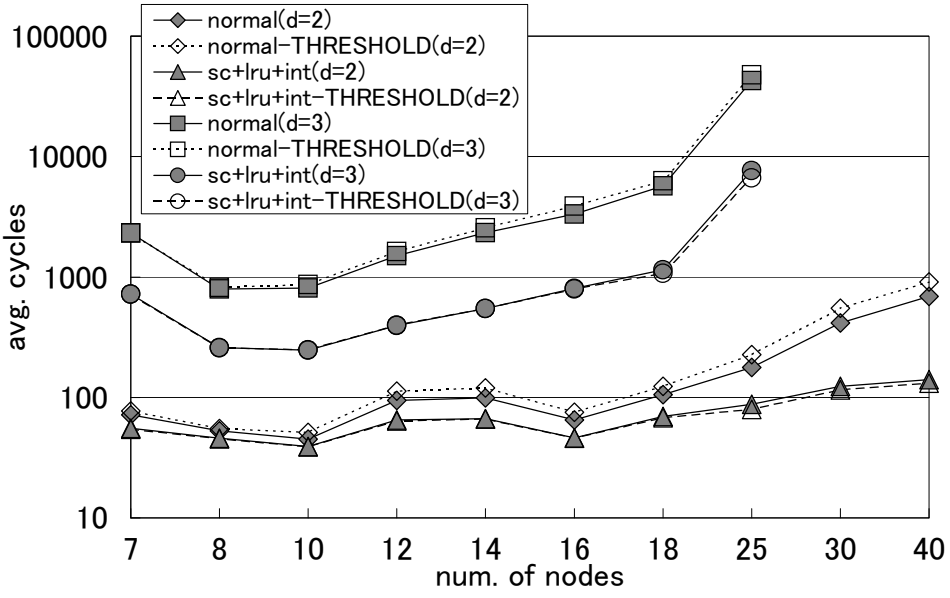


図 3.8: 変数値配信パス変更の影響 (制約重み=1,d=2,3)

3.7.4　変数値配信パスの変更による影響

以上の手法では変数値の配信において，**VALUE** メッセージによる近傍への配信と，**COST** メッセージによるその他のノードへの配信 (2.5-A) が併用されている．このうち，**COST** メッセージの *context* の代わりに **THRESHOLD** メッセージの *context* により値を配信するように変更した場合の結果を図 3.8 に示す．いずれの場合でもサイクル数には大きな差は無い．これは，探索処理全体の効率において，深さ優先探索木によるコスト計算の方が影響が大きいためであると考えられる．

3.7.5　MPI 環境における評価

非同期分散環境でのアルゴリズムの挙動を確認するために MPI 環境における実験を行った．ただし，本実験で使用したプログラムは，シミュレーションで使用したプログラムを各ノードごとにプロセスとして実行するように変更し，送受信部に MPI を適用したものであり，処理の高速化やメッセージ受信のタイミング等に調整の余地がある．制約の重み $\{1, \cdots, 10\}$,d=3 の問題について，各ノード数ごとに任意の 1 つの問題を選択し，各 10 回の試行の結果を平均した．実験で使用した計算機の主な構成は Intel Xeon 1.8GHz(HyperThreading),512MB メモリ,100Mbps Ethernet, Windows2000,MPICH1.2.5 である．1 台の場合の最大サイクル数の平均を図 3.9 に示す．また，各結果の分散を表 3.5 に示す．結果には負荷変動，非同期性等の影響が見られるが，非同期分散環境においてもシミュレーションの結果とほぼ同様の特性が得られた．実行時間を表 3.4 に示す．実行時間については各提案

表 3.4: MPI 環境における実行時間 [ms](制約重み=$\{1, \cdots, 10\}$,d=3)

num. of nodes	method	1 processor	4 processors	speed up rate
8	normal	2670	1841	1.5
	shortcut	2934	1769	1.7
	sc+lru	960	625	1.5
	sc+lru+int.	488	278	1.8
12	normal	10366	5245	2.0
	shortcut	11083	5047	2.2
	sc+lru	3380	1684	2.0
	sc+lru+int.	1425	570	2.5
16	normal	22860	10208	2.2
	shortcut	20789	9144	2.3
	sc+lru	6603	3227	2.0
	sc+lru+int.	3174	1450	2.2
20	normal	128135	56606	2.3
	shortcut	81791	30958	2.6
	sc+lru	30674	13208	2.3
	sc+lru+int.	12086	5041	2.4
24	normal	88595	38206	2.3
	shortcut	65573	25033	2.6
	sc+lru	22491	9511	2.4
	sc+lru+int.	11128	4089	2.7
32	normal	895219	343636	2.6
	shortcut	303739	107809	2.8
	sc+lru	144059	53377	2.7
	sc+lru+int.	104459	35062	3.0

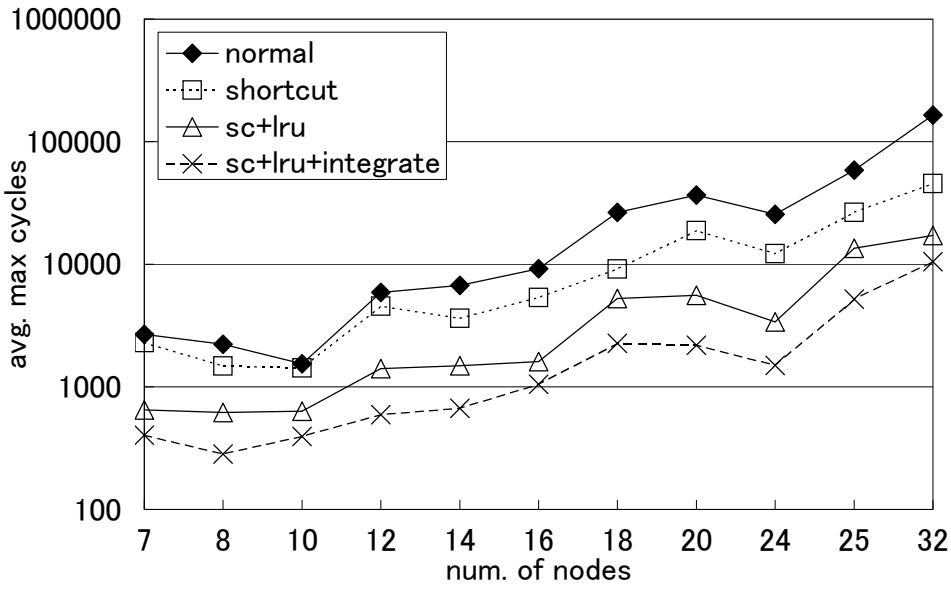


図 3.9: MPI 環境における最大サイクル数の平均 (制約重み=$\{1, \cdots, 10\}$,d=3,1 台)

表 3.5: 各結果の分散 (MPI 環境, 制約重み=$\{1 \cdots 10\}$,d=3, ノード数 16)

method	1 processor	4 processors
cycle		
normal	20074778	
shortcut	2649430	
sc+lru	119562	
sc+lru+int.	208107	
execution time		
normal	8272406	89895
shortcut	2340838	66134
sc+lru	128600	5802
sc+lru+int.	158003	38753

手法でサイクル数減少の効果による改善が得られた．1 台と 4 台の場合の比較において従来手法とほぼ同等の台数効果が得られ，提案手法による並列性への大きな影響は示されなかった．

3.8 まとめ

本章では，深さ優先探索木を基礎とした非同期分枝限定法に基づく，制約最適化の効率化手法を提案した．実験結果により提案手法によるバックトラックおよび再探索におけるオーバヘッド削減の効果が得られた．

本研究では，従来手法と同様に，基礎的な検討として，各ノードは単一の変数に対応し制約は 2 項制約となる問題に議論を限定した．これを多項制約に一般化することは容易であるが多項制約では制約密度は増加する．制約密度が高い問題であるほど，深さ優先探索木の木構造は線形になり並列性は損なわれる．従ってバックトラック効率の改善はより重要である．一方で，大規模で複雑な問題では，ユーザのパラメータにより許容されるコストで探索を打ち切る [15] ことが現実的である．その場合でも本手法のような効率改善手法は有効であると考えられる．また，各ノードが複数の変数を持つ場合については，ノード内部と外部で探索を多段化する手法 [26] の応用が考えられる

提案手法は基本的にヒューリスティクスに基づく探索効率の改善手法であり，最悪の場合の計算複雑度，および空間複雑度の基本的な性質は従来手法と同様である．実際的な場合において効率改善手法が有効な問題の規模の検討は今後の課題である．

検討の対象としたアルゴリズムは深さ優先探索木の存在を前提としているが，非同期分散探索における単調性の基礎としての木構造は本質的に重要であり，ブロードキャストに基づく対称なノードによる解法を構築する場合でもノードの順序付けの解決等に関連するものと考えられる．より効率的な解空間の学習手法の導入，他の解法との比較，実際的な問題への適用は今後の課題である．

第4章　深さ優先探索木に基づく分散制約最適化手法の動的問題への適用

4.1　はじめに

本章では，第 3 章で議論した分散制約最適化手法 [15] にもとづく実際的な分散システムを実装するための基礎的な検討として，動的環境への適用するための基本的な処理の枠組みを示す.

分散制約最適化問題の解法として，制約網に対する深さ優先探索木による変数の順序順序けを用いた手法が提案されている [15]．深さ優先探索木による順序付けでは，異なるサブツリーに配置された変数を持つノード (エージェント) 間に制約辺が無いため，問題の並列性を利用し探索を効率化できる．また, 厳密な深さ優先探索木では無いが，同様の性質を持つ順序付けを与える木をボトムアップに生成することもできる [14]．このような木は制約網に対する深さ優先探索木と同様に用いることができる.

制約充足/最適化問題の動的な問題への拡張として，制約充足/最適化問題の系列を順に解く，動的制約充足/最適化問題が提案されている [30] [31] [32] [33] ．一

方，分散制約充足問題 (DCSP), 分散制約最適化問題 (DCOP) を動的環境への適用するためには，解の探索のみではなく，問題の観測，解の決定を含めた全体的な処理を分散アルゴリズムによって構成し，それらを動的な問題の変化に追従させる必要がある．しかし，DCSP,DCOP に関する研究の多くでは，探索の前後処理について，分散スナップショットや大域的停止状態アルゴリズムなど [34] [35] [36] [37] の応用を仮定している．

一般に，分散処理のための基本的な構造として生成木は重要である．深さ優先探索木を用いた DCOP の解法に探索の前後処理を統合することは，動的な問題に追従する分散処理の枠組みを構成する上で重要であると考えられる．

本研究では，分散制約最適化手法の Adopt[15] を基本とし，局所的な知識を持つノード (エージェント) が，(1) 観測した制約を関連するノードに通知して制約網を再構築し，(2) 制約網に対する擬似的な木を構築し，(3) 生成された木に基づき分散制約最適化手法によりり解を探索し決定する，処理を反復し動的な問題に追従する基本的な枠組みを提案する．上記の各処理はボトムアップな非同期分散処理として統合される．処理はいずれかのノードで観測される制約の変化を起点として開始され，(1) では各ノードは制約網のスナップショットを記録し，安定状態となる．(2) では (1) の制約網に対する木の生成が大域的停止検出を兼ねる．(3) では (2) で生成された木に基づいて探索処理が実行され，その結果は 2 相コミットメントにより同期される．動的に変化する問題を扱うために，論理時刻を用いて，異なる問題に対する処理を識別する．また，解の探索処理については，前回の結果の再利用や，最良解の確保を優先する探索など，動的問題のために必要な処理を追加する．

以下では，先ず 4.2 で本章で対象とする動的な分散制約最適化問題について述べる．4.3 で提案手法の概要について述べる．4.4,4.5,4.6 でそれぞれ，制約の観測と制約網の構築，制約網に対する擬似的な木の構築，深さ優先探索木を用いた分散探索アルゴリズムの適用について述べる．4.7 では計算機実験による評価を示す．4.8 で本章の内容をまとめる．

4.2 動的な分散制約最適化問題

動的制約充足/最適化問題は，制約充足/最適化問題の系列として表され，その解法は問題の系列を順に解く．同様に，本研究では初期の検討のための例題として，動的な分散制約最適化問題を，DCOP の系列からなる問題として形式化する．

システムはメッセージ通信を行う複数のノード (エージェント) から構成される．ノードの集合を A とする．各ノード $i \in A$ は変数 x_i を持つ．変数 x_i は，x_i の値域 D_i の値 d_i をとる．x_i の値はノード i のみが変更する．以下では必要に応じて，ノード i と変数 x_i を区別せずに用いる．

問題の系列を $P = \{p_0, \cdots, p_s, \cdots\}$ で表す．$p_s = (C_s, F_s)$ であり，C_s は制約の集合，F_s は制約に対する評価関数の集合である．現在の問題 p_s について，x_i は他ノード j の変数 x_j と 2 項制約 $c_{i,j} \in C_s$ により関係する．制約 $c_{i,j}$ に対応する評価関数 $f_{i,j} \in F_s$ により，変数値の割り当て $\{(x_i, d_i), (x_j, d_j)\}$ についてのコストが評価される．大域コストは全ての制約に対するコストの評価の総和であり，大域コストを最小化する変数値の割り当てが最適解である．問題 p_s はある期間の経過後に次の問題 p_{s+1} に変化する．問題が変化するまでに最適解または準最適解を得る

必要がある．

各ノードは任意の他のノードとの間に FIFO のメッセージ通信リンクを持つ．異なる通信リンク間のメッセージの順序は保存されない．また，ノードおよび通信リンクの故障は考慮しない．

各ノードは大域的に無矛盾な変数 (ノード) 名およびその値域についての知識を持つが，各制約は関連する 2 ノードまたはいずれか 1 ノードが観測するものとする．各ノードは問題を解く前に制約で関連するノードと無矛盾な制約の知識を得る必要がある．

本研究では初期の検討として，ノード数と制約密度が比較的小規模な問題を対象とする，また，準最適解を得られる程度の時間間隔で問題が変化するものとし，問題が変化するまでの時間ついての知識は利用しない．このような例として，センサ網など分散処理環境の資源割り当て処理において，通常は定期的に問題が変化するまでに最適解が得られる規模の問題を扱い，負荷や問題の変化が一時的に増加した場合には，可能な近似解を利用することなどが考えられる．

4.3　動的な問題に追従する枠組み

本研究では，分散制約最適化手法である Adopt[15] を基礎とし，動的に変化する問題に対して，制約の観測，制約網の構築，変数の順序付け，解の探索・決定，の一連の反復処理を，ボトムアップな非同期分散処理として統合するための，基本的な手法を提案する．

処理の概要を図 4.1 に示す．(a) 各ノードは自ノードの関連する制約の変化を観

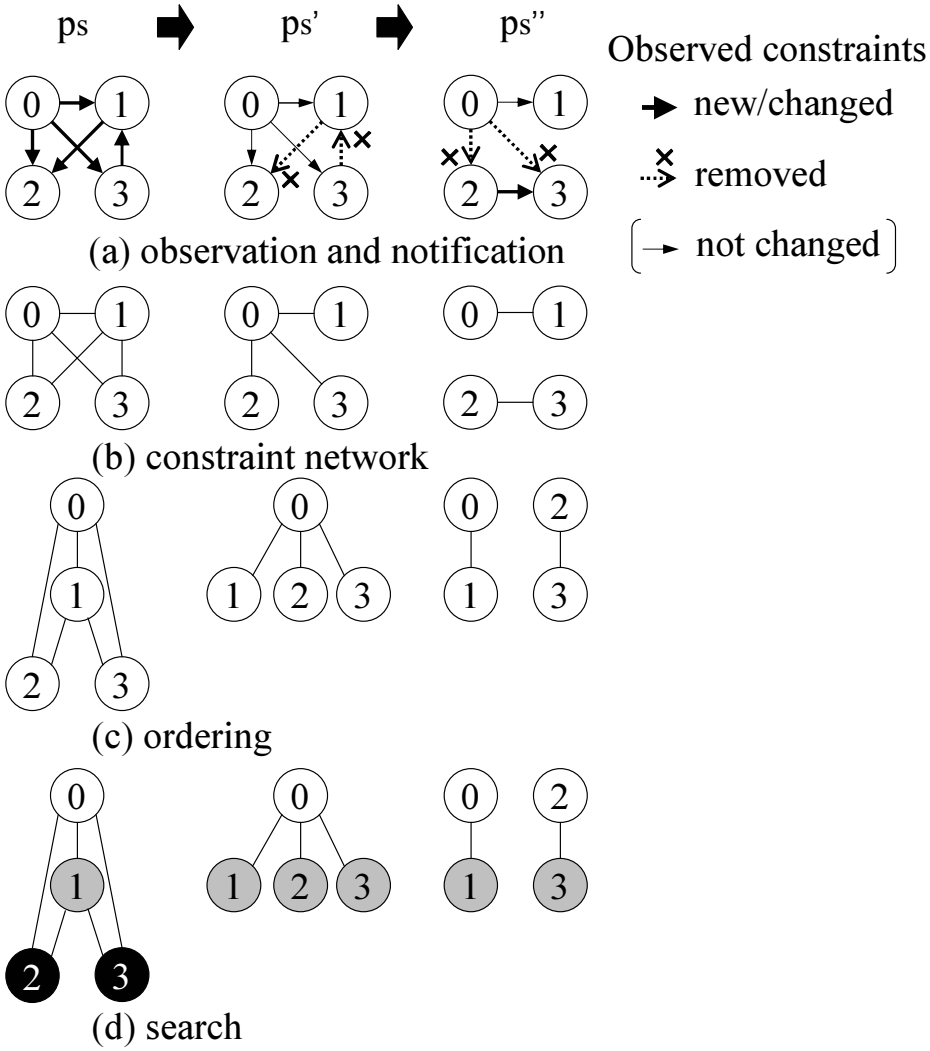

ps
ps'
ps''
Observed constraints
new/changed
removed
not changed
(a) observation and notification
(b) constraint network
(c) ordering
(d) search

図 4.1: 処理の概要

測し，制約で関連するノードへと通知する．(b) 制約の通知により，制約の更新および近傍ノードの追加・削除が行われ，制約網が再構築される．(c) 制約で関連するノードは制約網に対する擬似的な木によるノード (変数) 順序を生成する．(d) 順序付けに基づいて，解を探索・決定する．

問題の変化により制約網が複数の連結成分に分離する場合や，複数の連結成分が結合する場合は，上記の処理は分離・統合した各連結成分に属するノードの集合ごとに実行される．

このような，処理を非同期分散処理として構成するために，提案手法では，全体の処理を 3 つの階層により表す．各ノード i の処理の概要を図 4.2 に示す．ここでは，動的な問題を扱うために，論理時刻 clk により複数の問題に対する各層の処理を識別する．上位層の処理は同一論理時刻の下位層の情報に依存する．最下位層では自ノードが制約を観測し，問題の変化を制約で関連するノードへ通知し制約網を再構築する．各ノードは制約網の再構築の収束を局所的に検出する ($Sts_{i,clk}$)．構築された制約網について，中間層では各ノードはボトムアップに，擬似的な木によるノード順序を生成する ($Tree_{i,clk}$)．最上位層では生成されたノード順序に基づいて解を探索する ($Adpt_{i,clk,0}$)．解の探索ではある程度の実時間特性を意図し，適当なスケジュールにより暫定的に解を決定する．さらに，最適解が得られるか問題が変化するまで，同一の問題について探索を行い解は段階的に改善される ($Adpt_{i,clk,1}, Adpt_{i,clk,2}, \cdots$)．

以上の処理では，後述の解の決定中の場合を除き，各ノードは問題の変化に従って現在の問題に対する処理を破棄し，次の問題に対する処理を開始できる．以下では各層の処理について示す．

4.4 制約の観測と制約網の構築

各ノードは自ノードの関連する制約の変化を観測したとき，制約で関連する相手ノードに通知し制約網を再構築する．この処理は，論理時刻を用いたスナップショットアルゴリズムに基づく手法を用いる．

処理の概要は次の通りである．(1) 各ノードは自ノードが関連する制約の変化を観測したとき，その制約で関連する相手ノードに通知する．(2) 各ノードは観測および通知された制約を統合し，制約で関連するノードを近傍ノードとする．(3) 各ノードは論理時計を持ち，自ノードまたは近傍ノードの状態が変化したとき，論理時刻を増加する．論理時刻は近傍ノードにも通知される．(4) 各ノードは自ノードと全ての近傍ノードの論理時刻が等しいとき，問題の変化が局所的に収束したことを検出する．

観測される制約の変化によっては，制約網が複数の連結成分に分離する場合がある．このとき，上記の処理は各連結成分ごとに独立して実行される．

4.4.1 各ノードの状態の管理

ノード i が現在知る制約と関連ノードの情報を Sts_i で表す．Sts_i は次の要素からなる．

- C_i^{out}, F_i^{out} : 観測した制約, 評価関数の集合
- C_i^{in}, F_i^{in} : 受信した制約, 評価関数の集合
- Deg_i^{in}, Clk_i^{in} : 受信した次数, 論理時刻の集合

- C_i, F_i：制約, 評価関数の集合
- Nbr_i：近傍ノード名の集合

以下では表記を簡潔にするため，各集合の要素を添え字により識別する．

C_i^{out},F_i^{out} はそれぞれノード i が観測した制約, 評価関数の集合である．$c_{i,j}^{out} \in C_i^{out}$, $f_{i,j}^{out} \in F_i^{out}$ の変化は相手ノード j に通知されなければならない．起こりうる変化は，(1)$c_{i,j}^{out}, f_{i,j}^{out}$ を新たに観測するかその変化を観測する，(2)$c_{i,j}^{out}, f_{i,j}^{out}$ が観測されなくなる, のいずれかである．

C_i^{in},F_i^{in} はそれぞれノード i が他ノードから受信した制約と評価関数の集合である．観測および受信により，$C_i = C_i^{out} \cup C_i^{in}$, $F_i = F_i^{out} \cup F_i^{in}$ を得る．C_i^{out}, F_i^{out} の要素と C_i^{in}, F_i^{in} の要素が重複する場合は，それらを相手ノードと矛盾の無い方法で統合する[1]．

Nbr_i は近傍ノード名の集合であり．C_i に含まれる制約によりノード i と関連する全てのノード名を含む．次数 $|Nbr_i|$ の変化は $j \in Nbr_i$ に通知されなければならない．Deg_i^{in} は $j \in Nbr_i$ から受信した次数の集合であり，後述の木の構築処理で参照される．

各ノード i は論理時計を持ち，その論理時刻 clk_i により現在の状態を識別する．clk_i は他ノードへ情報を通知する際に同時に通知される．ノード i が $j \in Nbr_i$ から受信した論理時刻の集合を Clk_i^{in} で表す．clk_i は $clk_i \leftarrow \max(clk_i + \alpha, \max_j clk_j \in Clk_i^{in})$ により増加される．ただし，α は $C_{i,j}^{out}$,$F_{i,j}^{out}$,$|Nbr_i|$ に変化があるとき 1，それ以外のとき 0 である．

[1]本研究の例題では，制約で関連する 2 ノードが重複して同一の制約を観測する場合，メッセージの遅延を無視すれば矛盾は生じない．

4.4.2 関連ノードへの状態の通知

ノード i の状態が変化し clk_i が更新されたとき，ノード i は制約で関連するノードに，**STS** メッセージにより，制約の変化，次数，論理時刻を通知する．**STS** メッセージの送受信の手続きを図 4.3 に示す．本研究では，図 4.3 の (1)〜(4) に整理した．メッセージ中の () は要素の省略を示し，受信手続きでも省略された要素の処理をは行わない．

STS 受信では，メッセージの遅延により，Nbr_i に含まれないノードから (1) 以外のメッセージを受信した場合，そのメッセージは無視される．

4.4.3 局所的な安定状態の検出

制約の変化が観測されなくなれば，**STS** メッセージの通知は収束する．このような状態を各ノードは局所的に検出する．clk_i と Clk_i^{in} に含まれる論理時刻が全て clk に等しいとき，ノード i は clk で局所的に安定であるとする．このときの Sts_i を clk における部分的な問題 $Sts_{i,clk}$ として識別する．clk で局所的に安定なノード i の近傍ノード j が，clk で局所的に安定するためには，clk までの i からの **STS** メッセージを受信しなければならないため，$Sts_{i,clk}$ と $Sts_{j,clk}$ は矛盾しない．

制約で関連する全ノードが局所的に安定したことの検出は，後述の処理で構築される木が制約網の連結成分の生成木とみなせるため，木の完成が検出を兼ねる．

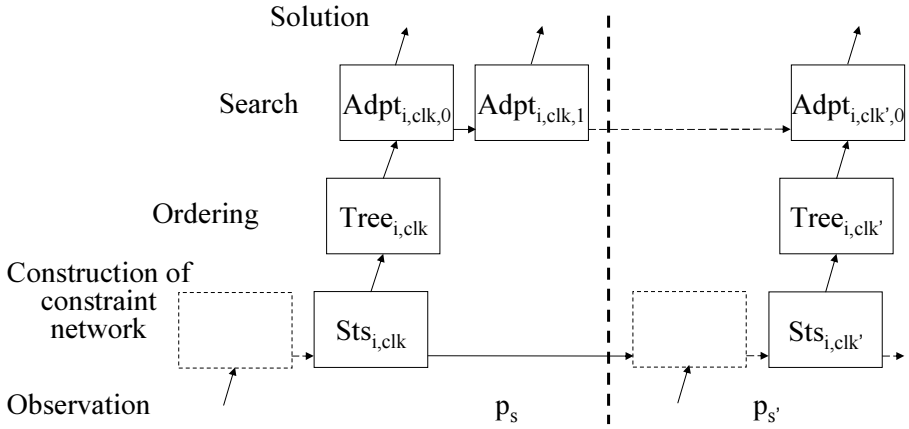


図 4.2: 処理の枠組み

---

(1) [(**STS**,($c^{out}_{i,j}$,$f^{out}_{i,j}$),(),$|Nbr_i|$,clk_i) **送信**]
送信する条件: $c^{out}_{i,j}$,$f^{out}_{i,j}$ が追加, 変更された
送信先: j

(2) [(**STS**,(),(),$|Nbr_i|$,clk_i) **送信**]
送信する条件:
メッセージ受信により $|Nbr_i|$,clk_i が更新された
送信先: $j \in Nbr_i$

(3) [(**STS**,(),(),(),clk_i) **送信**]
送信する条件:
メッセージ受信により clk_i が更新された
送信先: $j \in Nbr_i$

(4) [(**STS**,(),($c^{out}_{i,j}$,$f^{out}_{i,j}$),$|Nbr_i|$,clk_i) **送信**]
送信する条件: $c^{out}_{i,j}$,$f^{out}_{i,j}$ が削除された
送信先: j

[(**STS**,($c_{k,i}$,$f_{k,i}$),($c^{-}_{k,i}$,$f^{-}_{k,i}$),deg_k,clk_k) **受信**]
C^{in}_i,F^{in}_i に $c_{k,i}$,$f_{k,i}$ に追加・更新する
または $c^{-}_{k,i}$,$f^{-}_{k,i}$ を削除する
$deg_{i,k} \in Deg^{in}_i$ を deg_k に追加・更新する
$clk_{i,k} \in Clk^{in}_i$ を clk_k に追加・更新する

図 4.3: **STS** メッセージの送受信

4.5 擬似的な木の構築

ノード i はある論理時刻の問題 Sts_{i,clk_i} について，同じ論理時刻の問題で関連するノードとともに，ボトムアップに擬似的な木を構築する (図 4.4)．このような方法は [14] でも簡単に述べられているが，本研究では，提案手法の枠組みにおける下位および上位層の処理との統合方法を示す，また，停止検出条件を含めたアルゴリズムを示す．以下では処理の概要のみ述べ，詳細は章末の付録 4.A に示す．

ノード i は現在の論理時刻 clk_i に対する，木構造の情報 $Tree_{i,clk_i}$ を持つ．$Tree_{i,clk_i}$ は，入力される制約網に関する情報として Sts_{i,clk_i} の要素の複製である C_i,F_i,Nbr_i,Deg_i^{in} を持つ．また，出力される木に関する情報として，子ノード名の集合 $Chld_i$, 親ノード名 $parent_i$ を持つ．

各ノード i は Nbr_i と Deg_i^{in} にもとづいて，最大次数を優先する順序付けにより，Nbr_i を上位および下位ノードに分類する．その後，局所的な最下位ノードは親ノードを決定し，**TREE** メッセージにより，親ノードを含む上位ノードに，自ノードの情報を通知する (図 4.4(a))．上位ノードは，子ノードが全て揃ったとき，親ノードを決定し，上位ノードに通知する (図 4.4(b))．同様の処理を根ノードまで反復する (図 4.4(c))．この手法により得られる木では, 図 4.4(c) のノード 1,2 間に制約辺のように，制約辺が存在しない親子ノード間に制約辺が挿入される場合がある．この制約辺はコストが常に 0 であり次数の評価には含まれない．

ある論理時刻 clk について，木の要素であるノード k が $Tree_{k,clk}$ に対応する $Sts_{k,clk}$ を一度も持たない場合，その木は完成しない．各ノード i は木の構築の途中であっても clk_i の増加に伴って $Tree_{i,clk_i}$ を破棄し，次の論理時刻の木の構築に移行できる．

4.6　分散制約最適化手法の適用

解の探索処理は分散制約最適化手法の Adopt[15] に基づく．この手法では，事前に深さ優先探索木に基いて順序付けされたノードが，分枝限定法を基礎とする非同期分散探索を行う．以下ではアルゴリズムの概要のみ示す．また，メッセージの流れを図 4.5(a)(b) に示す.

Adopt の処理は，最適コストの探索と，解の決定，の 2 つの段階から成る．最適コストの探索では，(1) 各ノード i は，i を根とするサブツリーについての解のコストの下界値が，i に配分されたコスト値 $threshold_i$ を下回るように自変数の値 d_i を選択する．d_i は制約で関連する下位ノードに同報される (**VALUE** メッセージ). (2) 各ノードはサブツリーについての解のコストの上下界値を集計し，親ノードに通知する (**COST** メッセージ)．また，$threshold_i$ を上下界値を超えないように制限する．(3) 各ノード i は $threshold_i$ を，自ノードと各子ノード以下のサブツリーに配分し，各子ノードに通知する (**THRESHOLD** メッセージ).

以上の動作を反復し，根ノード r では $threshold_r$ と解のコストの上界値が一致する．このとき，アルゴリズムは解の決定の段階に移行する．根ノードは自変数の値を最適解に固定し子ノードに終了を通知する (**TERMINATE** メッセージ)．子ノードは同様に最適解を探索・決定し，さらに子ノードに終了を通知する．最終的にアルゴリズムは大域的に終了する．また，特に根ノードで解のコストの上下界値が一致したとき，最適コストであることが検出できる．以下では，提案手法において，Adopt を動的な問題への適用するための拡張方法について示す．また，以下の内容に関して，解の決定の同期，探索処理の開始の条件，メッセージの受信，については章末の付録 4.B で補足する.

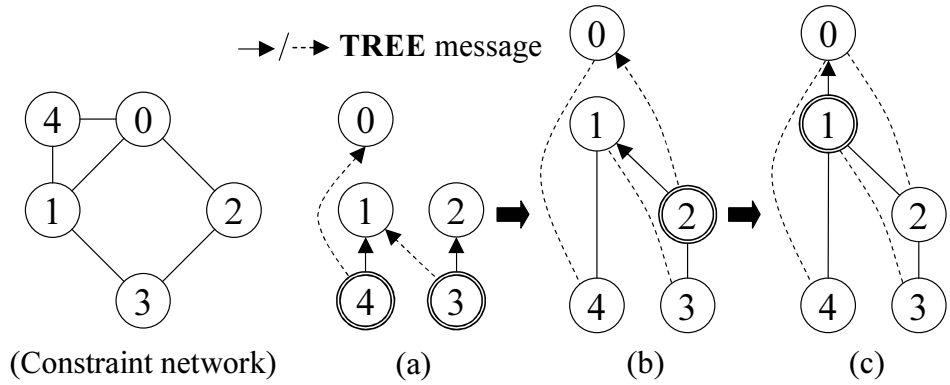


図 4.4: 木の生成

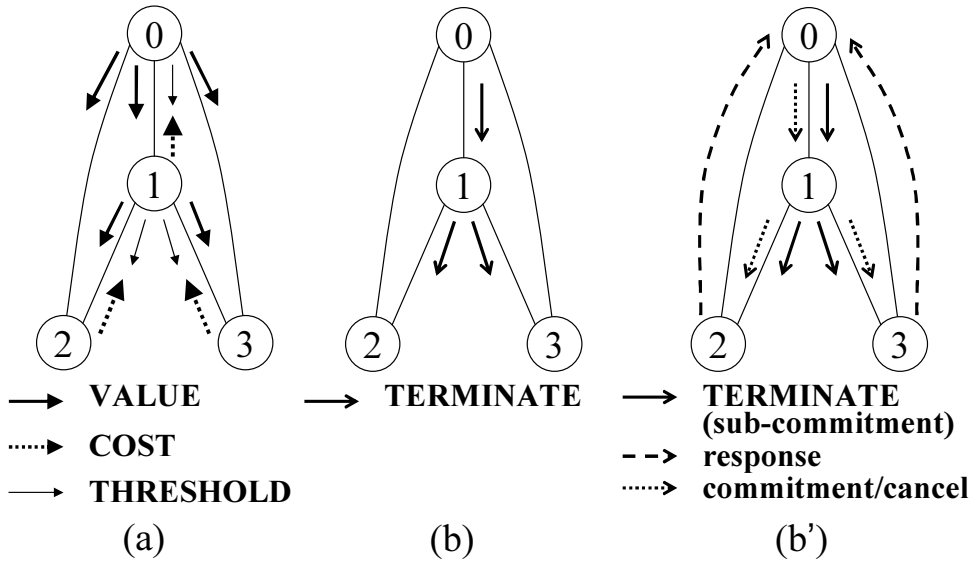


図 4.5: Adopt のメッセージ

4.6.1　論理時刻と解の番号による探索処理の識別

提案手法では，ノード i の解の探索処理は現在の論理時刻 clk_i の木 $Tree_{i,clk_i}$ について実行される．clk_i が増加したとき，各ノードはそれまでの木と探索処理を破棄し，新しい木の構築および探索処理を準備する．

動的な問題の解法には，実時間性や Anytime アルゴリズム [38] の性質があることが望ましい．しかし，Adopt は，基本的にはバックトラックに基づく下界優先探索手法であり，その精確な探索時間は不明である．そこで，適当な探索スケジュールに基づき，あるコストの上界値が根ノードで得られた時，コストの探索を中断し解の決定に移行する．解の決定後に問題が変化していなければ，次回の探索処理で解の改善を試みる．論理時刻 clk の問題について，複数回の探索処理を実行するため，何回目の探索処理であるかを番号 $slt = 0, 1, \cdots$ により表し，clk, slt のペアにより探索処理を識別する．また，Adopt の各メッセージに clk, slt の情報を追加し，どの探索処理に対応するメッセージであるかを識別する．

ノード i の clk,slt についての，探索処理の情報を $Adpt_{i,clk,slt}$ で表す．$Adpt_{i,clk,slt}$ は，Adopt のための情報に加えて，$Tree_{i,clk}$ の要素の複製 C_i, F_i, $Chld_i$, $parent_i$ および Nbr_i から導出される上下近傍ノードの情報を持つ．

4.6.2　解の決定の同期

上述のように，論理時刻 clk_i が増加したとき，それまでの探索処理は破棄される．しかし，解の決定は，木を構成する全てのノードについて無矛盾な解を得るために，同期されなければならない．特に最適解付近の解の決定では比較的多くの

探索サイクル数を要する場合があり，この間に問題が変化する場合を考慮する必要がある．そこで，解の決定に 2 相コミットメントに基づく手法を導入する．(図 4.5(b'))．解の決定の 1 相目で全ての葉ノードまで解を確保した後，2 相目で解を決定する．1 相目の途中で，根ノードの論理時刻が増加したとき，確保した解は 2 相目で取り消される．

4.6.3 複数の問題の確保と前回結果の再利用

解の決定中に下位ノードの処理待ちとなるノードは，次の問題の探索処理に備える必要がある．また，動的な問題では前回の探索結果など再利用することが必要になる．そこで，ノード i の $Adpt_{i,clk,slt}$ について，現在の探索中の $Adpt_i^{cur}$, 解の決定中の $Adpt_i^{sub}$, 前回の探索結果の $Adpt_i^{cmt}$ の 3 つを用いる (図 4.6)．

$Adpt_i^{cmt}$ の応用については, 次回の探索の効率化 [31]，解の安定性の確保 [33] などが考えられるが，さらに問題のクラスの特定や手法の拡張が必要である．本研究では，簡単な例として以下を用いた．(1) $Adpt_i^{cur}$ における変数値 d_i^{cur} は $Adpt_i^{cmt}$ の d_i^{cmt} を初期値とする．これは特に，問題が変化したときに，それまでの解を改善できる機会を得ることを意図する．(2) 根ノード r において，問題の変化が無く，前回の解を改善するとき，$Adpt_r^{cur}$ の $threshold_r^{cur}$ の初期値を $Adpt_r^{cmt}$ の解のコストの上下界値にもとづいて決定する．また，問題が変化した場合には $threshold_r^{cur}$ は十分に大きな値に初期化する．これは近似解の許容誤差を与える Bounded-Error[15] を段階的に小さくすることに類似する．

4.7　評価

計算機実験により提案手法を評価した．動的な分散制約最適化問題の解法の，総合的な効率の議論は簡単では無いと考えれる．本研究では初期の検討として，3 色のグラフ彩色問題 [15] の系列を順に解く例題について，提案手法を適用した結果を評価した．

各ノード間のメッセージ通信はシミュレーションにより模倣し，以下を反復するサイクル動作を処理の単位とした．(1) 各ノードは送受信キューを持ち，受信キューのメッセージを処理し，送信キューにメッセージを書き込む．(2) 各ノードの送信キューのメッセージを送信先の受信キューに移す．

各系列の問題を順に解き，各問題が次の問題に変化するまでに得られた最適解の数を評価した．ノード数 $n = 10$〜40 に対し，制約辺の密度 $d = 2, 3$ により，$n \cdot d$ の制約辺を持つ問題をランダムに生成した．$d = 2$ では充足可能な問題が多く，$d = 3$ では充足不能な問題が多い．各問題の規模を同一にするために，制約網が単一の連結成分を持つ問題を用いた．各制約の重さは 1(最大制約充足問題) とした．各制約はいずれか片方のノードに観測されるものとした．問題の変化として，各問題は，サイクル数 T が経過した時点で次の問題に変化するものとした．全体の 10% の制約が削除され，同数の制約が追加されるものとした．

問題が変化していないときに解を改善するための，根ノード r の $Adpt_r^{cur}$ の $threshold_r$ の初期値は，($Adpt_r^{cmt}$ におけるコストの上界値) $-\ \Delta$ により決定し，$\Delta = 1$ とした．

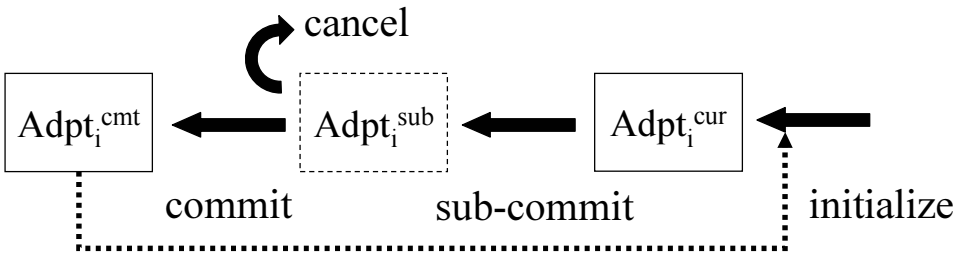


図 4.6: 解の決定の同期

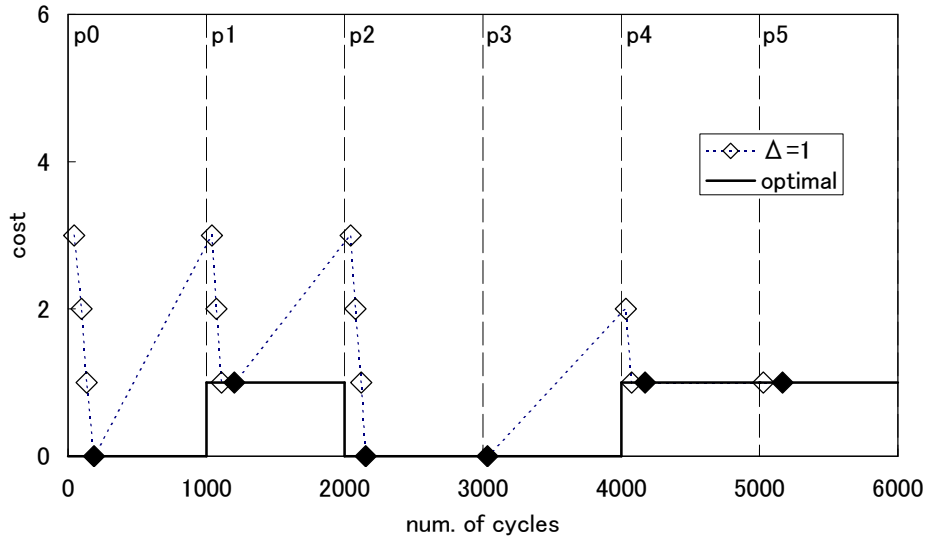


図 4.7: 解のコストの例 (n=20,d=2,T=1000), 各問題 $p0$〜$p5$ の◆において最適解であることを検出した.

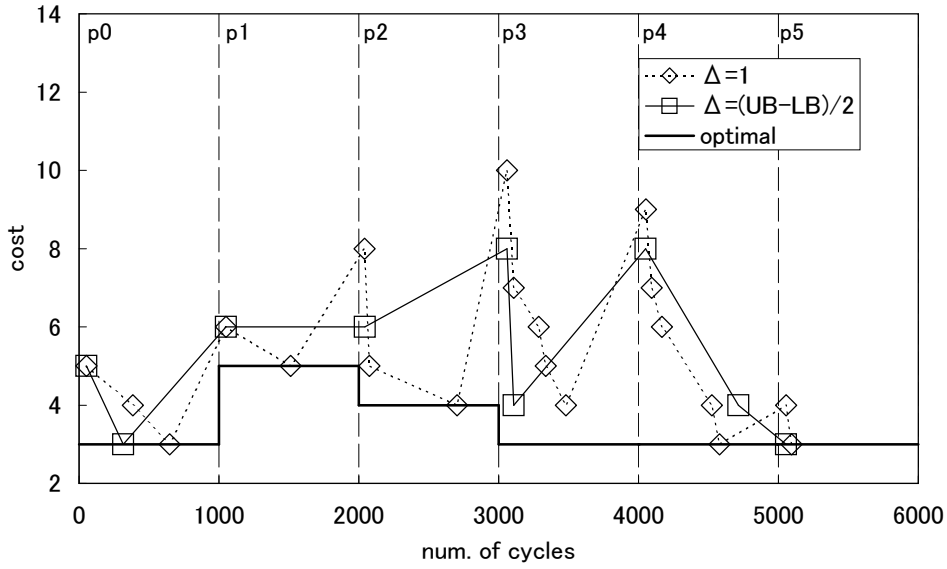


図 4.8: 解のコストの例 (n=20,d=3,T=1000)

表 4.1: 最適解の数 ($d = 2$, 各 600 問における割合 (%))

method	n	T	
		500	1000
dyadpt+	10	100	100
	20	100	100
	30	100	100
	40	99.3	100
dyadpt	10	100	100
	20	100	100
	30	87.7	97.2
	40	76.5	90.0
lnradpt	10	99.2	100
	20	27.2	44.5
	30	1.8	4.7
	40	0.3	0.5

表 4.2: 最適解の数 ($d = 3$, 各 600 問における割合 (%))

method	n	T			
		500	1000	2000	5000
dyadpt+	10	100	100	100	100
	20	84.0	98.5	100	100
	30	27.8	45.0	63.0	91.3
	40	4.8	9.5	12.7	24.2
dyadpt	10	100	100	100	100
	20	50.5	72.2	85.7	100
	30	11.2	16.3	26.3	38.8
	40	1.3	1.8	2.7	4.3
lnradpt	10	86.5	97.3	99.7	100
	20	9.3	11.3	16.2	25.8
	30	0.5	0.8	0.8	1.7
	40	0.0	0.0	0.0	0.0

表 4.3: 解のコストの最適解との誤差の平均

method	n	T			
		500	1000	2000	5000
dyadpt+	20	0.2	0.0	0	0
	30	1.4	1.0	0.6	0.1
	40	3.4	2.8	2.4	1.6
dyadpt	20	0.9	0.4	0.2	0
	30	2.6	2.3	1.9	1.4
	40	4.7	4.4	4.0	3.6
lnradpt	20	2.6	2.3	1.9	1.5
	30	5.3	5.1	4.8	4.5
	40	8.3	8.1	7.8	7.6

4.7.1　解の決定と得られた解のコスト

各問題について決定された解のコストの例を，図 4.7,4.8 に $n = 20, d = 2, 3, T = 1000$ の場合の例を示す．図 4.7 の d=2 の例では，問題 $p0$〜$p5$ とも，最終的に根ノードで解のコストの上下界値が一致し，最適解が得られたことを検出した[2]．ただし，問題 $p1, p4, p5$ では，解のコストが最適コスト (optimal) に等しくなった時点では，根ノードで得られた解のコストの下界値は上界値である最適コストより小さく，最適解を得たことは検出できなかった．これらの問題では，さらに次回の探索で最適解を検出した．

図 4.8 の d=3 の例では，コストの改善量が大きい場合としてパラメータ $\Delta =$ ($Adpt_r^{cmt}$における解のコストの上下界値の差)/2 とした場合 ($Delta = (UB - LB)/2$) も合わせて示す．特に充足不能な問題では，最適コストを下回る値を $threshold_r$ に設定した場合，Adopt は下界値の探索を優先し探索に時間を要する．これを避けるために，$threshold_r$ は最適コストを上回るようにすることが効率的であると考えられる．図 4.8 の例では，いずれの場合も根ノードにおいて解のコストの下界値は 0 であった。このため，解のコストが最適コスト (optimal) に等しくなった場合を含め，いずれの問題でもサイクル数 $T = 1000$ 以内で最適解を得たことは検出できなかった．$\Delta = (UB - LB)/2$ の場合は，$p0$ では $\Delta = 1$ より早く最適解を得た．一方で，$p1, p2$ では時間内に解が改善されなかった．これは $threshold_r$ が最適コスト以下に初期化されることによると考えられる．また，$p2$〜$p5$ では，$\Delta = 1$ の方が問題の変化直後の解のコストが大きいことなど，前回の探索結果による初

[2]Adopt では根ノードでコストの上下界値が一致したとき，そのコストが最適コストであることを検出できる．しかし，本研究では，適当な時期に探索を中断して解を決定する．このため，コストの上界値が最適コストと一致していても，下界値が最適コストより小さい場合がある．このような場合には最適コストが得られたことは検出されない．

期解が次回の探索に影響する傾向が見られる．

また，図 4.7 の $p0, p1, p2$ や図 4.8 の $p2, p3, p4$ では，$\Delta = 1$ の小刻みな解の決定は冗長である．解の決定についてはさらに，問題の規模，制約密度，問題の変化までの時間などに応じた検討の余地がある．

4.7.2 解のコストの評価

各問題において最終的に得られた解のコストについて評価した，n, d, T の組み合わせについての 10 系列を生成し初期解を変えて各 10 回の試行を行った．各系列に含まれる問題の数は 6 とした．

比較のために，提案手法 (dyadpt) および提案手法に 3 章で述べた Adopt に対する効率改善手法を適用したもの (dyadpt+) を用いた[3]．Adopt は比較的簡単なバックトラック処理に基づく手法であり，探索効率に改善の余地があるため，上記の効率化手法を併用した場合についても評価した．効率化手法では木の構築と Adopt の処理に簡単な拡張が必要となる．拡張方法の概要を章末の付録 4.C に示す．

また，適当な線形のノード順序を用い，提案手法と同様の規則で Adopt を反復実行した場合 (lnradpt) を比較した．この場合のサイクル数については，問題の観測からノードの順序付けまでの処理を無視し，探索および解の決定のみ評価した．

各試行に含まれる 600 問について最適解が得られた割合を表 4.1,4.2 に示す．解の探索処理の効率は，最適解を得るために必要なサイクル数において支配的である．効率改善手法を適用した場合 (dyadpt+) は，$d = 2$ の問題では $T = 1000$ まで

[3]ここでは，3 章の 3.7 における sc+lru+integrate と同様の各効率化手法を全て適用する手法を用いた．また，子ノードのコスト情報のキャッシュのサイズもと同様とした．

表 4.4: 問題毎の STS,TREE メッセージ数

n	d	STS			TREE		
		mean	min	max	mean	min	max
10	2	70	41	84	28	20	43
	3	118	61	126	37	31	49
20	2	159	83	168	77	55	106
	3	245	124	251	108	78	142
30	2	244	124	251	136	100	197
	3	369	363	375	203	162	255
40	2	328	168	333	206	159	276
	3	492	485	499	323	261	412

表 4.5: 問題毎の制約網, 木の構築の収束までのサイクル数

n	d	STS			STS+TREE		
		mean	min	max	mean	min	max
10	2	5	3	7	10	7	14
	3	5	3	6	12	9	14
20	2	6	3	9	15	11	20
	3	5	3	7	18	13	23
30	2	6	3	9	19	14	28
	3	5	4	7	23	18	30
40	2	6	4	9	23	18	30
	3	5	4	7	29	24	36

に全ての場合で最適解が得られた．一方，$d = 3, n = 40$ の問題では $T = 5000$ でも最適解が得られない場合が多い．$d = 3$ の場合の解のコストの最適解との誤差の平均を表 4.3 に示す．解のコストの誤差についてもそれぞれ探索手法の効率に応じた結果となった．

4.7.3 制約網および木の構築

提案手法について各問題毎の **STS**，**TREE** メッセージ数を表 4.4 に示す．**STS** の平均のメッセージ数は，各ノードの近傍数の総和 $2 \times n \times d$ の 2 倍程度となった．**TREE** メッセージ数は，制約数，ノード数，生成される木の深さに依存すると考えられるが，**STS** メッセージ数より少ない程度となった．

また，各問題毎の制約網, 木の構築が収束するまでのサイクル数を図 4.5 にを示す．$n = 40, d = 3$ の場合でも木の構築の収束まで平均サイクル数は 30 程度であり，解の探索処理と比較して十分に小さい．

各問題毎の木の深さを図 4.6 にを示す．表 4.1,4.2 における，dyadpt の lnradpt の対する探索の効率化はこの木の深さに依存する．

4.7.4 問題あたりのメッセージ数

表 4.7 に提案手法 (dyadpt) の場合の問題あたりのメッセージ数を示す．提案手法における Adopt のメッセージ数の総合的な評価は難しいが，各探索処理では，従来の Adopt のメッセージ数 [15] と同様の傾向になると考えられる．しかし解の決定中は，木の上位のノードは処理待ちとなりメッセージ送受信を行わない．また，

図 4.7 の例のように，最適解が得られたことを検出した場合は，それ以降の探索処理は不要であるため．次に問題が変化するまでメッセージは送信されない．表 4.7 の $d = 2, n = 10, 20$ および $d = 3, n = 10$ では，最適解の検出による探索の終了のため，T を増加したときにメッセージ数の増加が収束している．

また，効率改善手法を追加した場合 (dyadpt+) は，**STS**,**TREE** メッセージ数は同数であるが，章末の付録 4.C に示す拡張により，木の構築の後処理のためにメッセージ数が $n - 1$ 増加する．また，Adopt のサイクルあたりのメッセージ数は，拡張される **COST** メッセージの影響で増加するが，その実際の増加量は，第 3 章の図 3.4 に示した結果と同様に従来手法の比較し十分少ない程度になる．

4.8　まとめ

本研究では，制約網にもとづく深さ優先探索木に基づく分散制約最適化手法である Adopt を動的な問題に追従させるための，基本的な処理の枠組みを示した．提案手法では，制約の観測と制約網の構築，変数の順序付け，探索および解の決定の同期，の処理をボトムアップな非同期分散処理として統合した．また，提案手法の有効性について計算機実験により評価した．

本研究では実装方法の初期の検討として，特に探索アルゴリズムと前後処理の統合方法について示した．提案手法における，前回の探索結果の再利用や，最良解の確保のためのスケジュールについては，さらに問題に応じた検討が必要である．前回の探索結果の履歴の応用としては，解の安定性を確保するための暫定制約 [33] の適用が考えられる．提案手法で基礎とした Adopt の探索の効率には課題

表 4.6: 生成された木の深さ

n	d	depth		
		mean	min	max
10	2	7	5	9
	3	8	6	10
20	2	11	8	16
	3	14	9	19
30	2	15	10	23
	3	20	15	26
40	2	19	13	26
	3	25	20	32

表 4.7: 問題あたりのメッセージ数

d	n	T			
		500	1000	2000	5000
2	10	1527	1527	1527	1527
	20	6555	7483	7465	7465
	30	12709	19407	25221	30984
	40	18506	30136	48397	71124
3	10	8437	14046	15583	15583
	20	15260	33979	72485	171373
	30	20463	45071	97091	261780
	40	25043	55027	118431	318143

があるが，さらに各種の効率化手法 [18] [17] の適用による改善や，問題の規模やクラスを制限することなどが考えられる．

また，本研究では簡単のために，各ノードが持つ変数は単一で，各変数が 2 項制約で関係する問題を扱ったが，多項制約の場合に拡張することは容易である．実際的には，各ノードが複数の変数を持ち，変数の構成や値域が変化する場合などの検討が必要である．これらを含めた応用的な問題への適用は今後の課題である．

4.A ボトムアップな木の構築処理

ノード i の論理時刻 clk に対する，木構造の情報を $Tree_{i,clk}$ で表す．$Tree_{i,clk}$ は次の要素からなる．

- C_i,F_i,Nbr_i,Deg_i^{in}：$Sts_{i,clk}$ の要素の複製
- Nds_i：関連ノード名の集合
- Deg_i：関連ノードの次数の集合
- Rel_i：関連ノードとそのサブツリーとの間の制約数の合計, の集合
- Nbr_i^u, Nbr_i^l：上位, 下位近傍ノード名の集合
- $Chld_i$：子ノード名の集合
- $parent_i$：親ノード名
- N_i^{leaf}：サブツリーの葉ノード数

このうち，C_i,F_i,Nbr_i,Deg_i^{in} が下位層からの入力情報である．また，C_i,F_i, および Nbr_i^u, Nbr_i^l,$Chld_i$，$parent_i$，N_i^{leaf} が上位層への出力情報である．

ノード i は現在の論理時刻 clk_i に対する $Tree_{i,clk_i}$ のみを保持する．一方で Sts_{i,clk_i} が得られるまで，C_i,F_i,Nbr_i,Deg_i^{in} は得られない．

ノード i の $Tree_{i,clk_i}$ についての処理を図 4.9 に示す．

ただし，Rel_i の要素は最初に参照されるとき 0 に初期化される．ノードの順序付けは最大次数ヒューリスティクスによる．ノード i, j の次数を deg_i, deg_j とするとき，$deg_i > deg_j$ であれば i は j より上位である．ただし,$deg_i = deg_j$ のときは，ノード

[**初期化**]
$Nds_i \leftarrow \phi$, $Deg_i \leftarrow \phi$, $Rel_i \leftarrow \phi$,
$Chld_i \leftarrow \phi$, $N_i^{leaf} \leftarrow 0$, (他の要素は未初期化).
[Sts_{i,clk_i} **が得られたとき**]
Nbr_i の要素を Nbr_i^u,Nbr_i^l に分類する.
if $Nbr_i^l = \phi$ **then** $N_i^{leaf} \leftarrow N_i^{leaf} + 1$ **endif**
$Nds_i \leftarrow Nds_i \cup Nbr_i$, $Deg_i \leftarrow Deg_i \cup Deg_i^{in}$.
$rel_{j,i} \in Rel_i$, $j \in Nbr_i^u$ について $rel_{j,i} \leftarrow rel_{j,i} + 1$.
[(**TREE,**$parent_i$**,**Nds_i**,**Deg_i**,**Rel_i**,**N_i^{leaf}**,**clk_i**) 送信**]
送信する条件: $rel_{i,i} = |Nbr_i^l|$
送信前の処理: Nds_i,Deg_i,Rel_i から
i および下位ノードの要素を削除する.
Nds_i の最下位ノードを $parent_i$ とする.
送信先: $j \in Nds_i$
[(**TREE,**$parent_k$**,**Nds_k**,**Deg_k**,**Rel_k**,**N_k^{leaf}**,**clk**) 受信**]
if $i = parent_k$ **then** $Chld_i \leftarrow Chld_i \cup \{k\}$,
$N_i^{leaf} \leftarrow N_i^{leaf} + N_k^{leaf}$.
$rel_{l,m} \in Rel_i$,$rel'_{l,m} \in Rel_k$ について
$rel_{l,m} \leftarrow rel_{l,m} + rel'_{l,m}$ **endif**
$Nds_i \leftarrow Nds_i \cup Nds_k$, $Deg_i \leftarrow Deg_i \cup Deg_k$.

図 4.9: $Tree_{i,clk_i}$ についての木の構築処理

名についての全順序関係が利用できるものとする．また，**TREE 受信**では，現在の論理時刻 clk_i より前の論理時刻のメッセージは無視される．

ノード i の子ノード以下のサブツリーには，i の下位近傍ノードが重複せずに一つ以上含まれる．全ての子ノードから **TREE** メッセージを受信したとき，ノード i と各サブツリーのノードとの間の制約数の合計 $rel_{i,i} \in Rel_i$ は，下位近傍ノード数に等しい．よって，i 以下のサブツリーが決定したことを，条件 $rel_{i,i} = |Nbr_i^l|$ の成立により判定できる．葉ノードではサブツリーは速やかに決定される．

4.B Adopt の処理の変更についての補足

4.B.1 終了処理の変更

現在の問題 $Adpt_i^{cur}$ についての終了処理は，図 4.10 のように変更される．**TERMINATE** メッセージには，根ノード名 x_{root}，根ノードにおいて最適解が得られているか否かを示すフラグ *opt* が追加される．

また，$Adpt_i^{sub}$ についての確定または取り消しのために，図 4.11 の処理が追加される．

4.B.2 実行開始の条件

$Adpt_{i,clk,0}$ の処理は，$Tree_{i,clk}$ 以下のサブツリーが決定するまで開始できない．$Tree_{i,clk}$ は下位ノードから順に決定されるため，$Adpt_{i,clk,0}$ の処理は下位ノードから順に開始できる．また，各ノードは，解のコストを正しく計算するために，全

[(**TERMINATE,** $\cdots$, x_{root},opt,clk, slt) **送信**]
送信する条件: Adopt の終了条件が成立した
$\wedge$ i は葉ノードではない
送信先: $x \in Chld_i$
送信後の処理:
$Adpt_i^{sub} \leftarrow Adpt_i^{cur}$, $Adpt_i^{cur} \leftarrow Adpt_{i,clk,slt+1}$
[(**TERMINATE,**$\cdots$, x_{root},opt,clk, slt) **受信**]
(Adopt の TERMINATE 受信処理)
x_{root}, opt を記録する
[(**TRM_OK,**clk, slt) **送信**]
送信する条件: Adopt の終了条件が成立した
$\wedge$ i は葉ノード
送信先: x_{root}
送信後の処理: (**TERMINATE 送信**と同様)

図 4.10: $Adpt_i^{cur}$ についての終了処理

[(**TRM_OK,**clk, slt) **受信**]
TRM_OK 受信数を記録
[(**TRM_COMMIT,**clk, slt) **送信**]
送信する条件: **TRM_COMMIT** を受信した
$\vee$ (i は根ノード $\wedge$ **TRM_OK** 受信数=N_i^{leaf})
送信先: $x \in Chld_i$
送信後の処理: $Adpt_i^{cmt} \leftarrow Adpt_i^{sub}$, $Adpt_i^{sub}$ を削除
[(**TRM_COMMIT,**clk, slt) **受信**]
TRM_COMMIT 送信
[(**TRM_CANCEL,**clk, slt) **送信**]
送信する条件: **TRM_CANCEL** を受信した
$\vee$ (i は根ノード $\wedge$ 論理時刻 clk_i が増加した)
送信先: $x \in Chld_i$
送信後の処理: $Adpt_i^{sub}$ を削除
[(**TRM_CANCEL,**clk, slt) **受信**]
TRM_CANCEL 送信

図 4.11: $Adpt_i^{sub}$ についての処理

ての上位近傍ノードから **VALUE** メッセージを受信するまで **COST** メッセージを送信しない．全ての上位近傍ノードから **VALUE** メッセージを全て受信したとき，親ノードは処理を開始していて **COST** メッセージを受信できる．

ノード i が $Adpt_{i,clk,slt}$ に対する **TERMINATE** メッセージを送信したとき，$Adpt_i^{sub} = Adpt_{i,clk,slt}$, $Adpt_i^{cur} = Adpt_{i,clk,slt+1}$ であり，次の問題の準備ができている．従って，$slt > 0$ のとき，**TRM_COMMIT** を送信したノード以下のノードは次の問題についてのメッセージを受信できるため，処理を上位ノードから開始できる．

解の決定中であり $Adpt_i^{sub}$ が記録されているノードは，$Adpt_i^{sub}$ の確定または取り消しまで，$Adpt_i^{cur}$ の処理を開始できない．しかし，他ノードから $Adpt_i^{cur}$ へのメッセージを受信する場合がある．メッセージの受信のみであれば自ノードの解の評価は不要であるため，アルゴリズムに矛盾は生じない．

また，既に最適解が得られているとき，それ以降の探索は冗長である．

以上より，$Adpt_i^{cur}$ に対する $Tree_{i,clk_i}$ 以下のサブツリーが決定していて，かつ最適解が得られていないとき $Adpt_i^{cur}$ に対する処理は開始される．ただし，$Adpt_i^{sub}$ が記録されているとき $Adpt_i^{cur}$ についてはメッセージ受信のみ行う．

4.B.3 メッセージ受信のタイミング

VALUE, **THRESHOLD**, **COST**, **TERMINATE** メッセージには論理時刻 clk と解の番号 slt を付加し，$Adpt_{i,clk,slt}$ についてのメッセージであることを示す．受信処理では $Adpt_i^{cur}$ より前のメッセージは無視される．ノード i の論理時刻 clk_i が増加したとき $Tree_{i,clk_i}$ と $Adpt_i^{cur}$ が更新されることと，上記の実行開始の条件により

$Adpt_i^{cur}$ より後のメッセージは受信されない．同様に，**TERM_OK**/ **COMMIT**/ **CANCEL** にも clk, slt が付加され, 受信処理では $Adpt_i^{sub}$ についてのメッセージ以外は無視される．

4.C　効率化手法の追加

3 章に示した効率改善手法は Adopt に，(1) 分枝限定法における backjumping に類するバックトラッキングの効率化，(2) 部分解とコストの学習による再探索の削減, を適用する．これらの拡張については 3 章で述べた手法に従った．特に上記 (1) では，従来の **COST** メッセージに加え，コストの下界のみを通知する **COST** メッセージが祖先ノードに送信される．そのため，各ノードは全ての祖先ノードとその順序を知らなければならない．そこで，木の完成後に，祖先ノードとその順序の情報を，根ノードから各葉ノードまで伝搬するメッセージを追加した．また，上記 (1) の効率化手法は，このメッセージの受信後に有効にした．以上の拡張では提案手法の非同期性を損なうような影響は無い．

第5章　擬似的な木の幅を考慮した動的計画法とバックトラックの統合

5.1　はじめに

本章では，第 3 章，第 4 章で議論の対象とした分散制約最適化手法の関連研究である，制約網に対する擬似的な木 (pseudo-tree) と動的計画法に基づく分散制約最適化手法 [46] について，擬似的な木の幅を考慮したメモリの制限のための基本的な手法を検討する．

近年，制約網に対する深さ優先探索木などの擬似的な木 (pseudo-tree) による変数順序付けを用いた分散制約最適化問題の解法が提案されている [16] [46] ．制約網に対する深さ優先探索木などの擬似的な木は，各部分木に配置された変数の間に制約辺が無いため，問題の持つ並列性を利用した探索の効率化が得られる．

このような擬似的な木を用いる，メモリ制限のある A*アルゴリズムに基づく非同期分散最適化手法が提案された [16]．この解法は問題のサイズに対する多項式の空間複雑度に制限し，非同期な分枝限定法に基づく探索を行う．一方でメッセージ数が多く，最悪の場合の計算複雑度は指数関数的である．

これに対し，動的計画法に特化した手法を適用することにより、木の深さについて多項式時間で解を得ることができる [45][46]．通信のオーバヘッドを考慮した場合，自由度の高い非同期バックトラックに基づく手法よりも，メッセージ数を削減できる動的計画法を主体とする手法は有効である．

しかし、このような手法の空間複雑度は，各ノードを根とするサブツリーから制約辺で関連する上位ノードの数によって示される，擬似的な木の幅に対して指数関数的に増加するため、適用可能な問題は制限される．制約密度が小さい問題に対して擬似的な木の幅は小さくなる傾向があるが、一般的な問題では木の幅を一定値以下にすることは厳密には難しい場合があると考えられる．

本論文では，動的計画法に基づく解法 [46] の補助的な手段として擬似的な木の幅が制限される値を超える部分について空間複雑度を削減するための手法を示す．一部の変数について部分的にバックトラックを導入する手法を示す．また，貪欲戦略の一つとして一部の変数の変数値を予め固定する手法を示す．

以下では，5.2 で対象とする分散制約最適化問題について述べ，5.3 で従来手法として，制約網に対する擬似的な木を用いた動的計画法に基づく分散探索アルゴリズムについて述べる．5.4 で提案手法として，従来手法に擬似的な木の幅を考慮した記憶のサイズの制限とバックトラックを導入する手法を示す．5.5 では計算機実験による評価を示す．5.6 では関連研究および考察を示す．5.7 で本章の内容をまとめる．

5.2 分散制約最適化問題

本研究で対象とする分散制約最適化問題について示す．システムはメッセージ通信を行うノード (エージェント) の集合から成る．ノードの集合を A とする．各ノード $i \in A$ は変数 x_i を持つ．変数 x_i は，x_i の値域 D_i の値 d_i をとる．以下では必要に応じて，ノード i と変数 x_i を区別せずに用いる．

C を制約の集合，F を制約に対する評価関数の集合とする．x_i は他ノード j の変数 x_j と 2 項制約 $c_{i,j} \in C$ により関係する．制約 $c_{i,j}$ に対応する評価関数 $f_{i,j} \in F$ により，変数値の割り当て $\{(x_i, d_i), (x_j, d_j)\}$ についてのコストが評価される．大域コストは全ての制約に対するコストの評価の総和であり，大域コストを最小化する変数値の割り当てが最適解である．

各ノードは任意の他のノードとの間に FIFO のメッセージ通信リンクを持つ．異なる通信リンク間のメッセージの順序は保存されない．また，ノードおよび通信リンクの故障は考慮しない．

5.3 従来手法 — 制約網に対する擬似的な木を用いた動的計画法

5.3.1 制約網に対する擬似的な木

制約網に対する深さ優先探索木などの擬似的な木 (pseudo-tree) は，各部分木に配置された変数の間に制約辺が無い性質があり，探索手法の効率化に利用できることが知られている [39] [40] [41] [43] [44] [16] [46] ．図 5.1 に制約網と擬似的な木

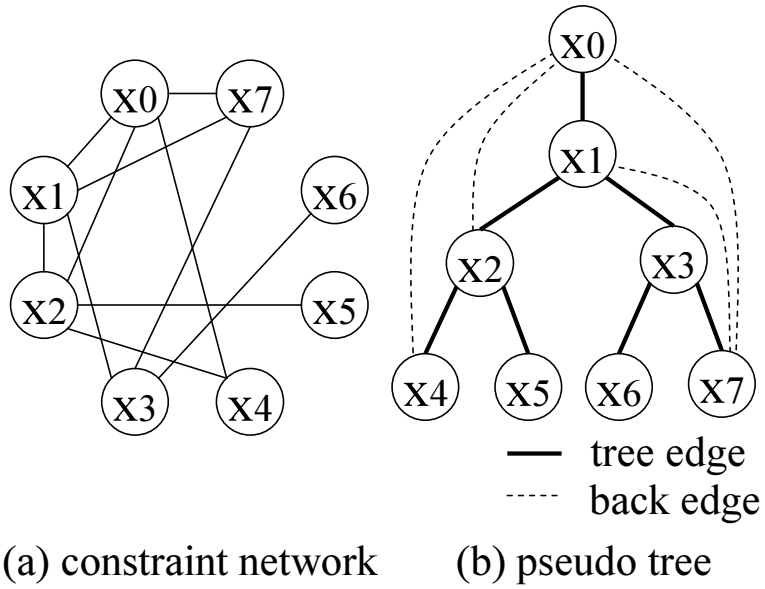


図 5.1: 制約網と pseudo tree

の例を示す．制約辺は，深さ優先木の枝となる木の辺 (tree edge) とそれ以外の後退辺 (back edge) に分類される．以下では，ノード x_i と制約辺で関連するノードを次の表記により示す.

- $P(x_i)$：木の辺で x_i と関連する親ノード
- $PP(x_i)$：後退辺で x_i と関連する祖先ノードの集合
- $C(x_i)$：木の辺で x_i と関連する子ノードの集合
- $PC(x_i)$：後退辺で x_i と関連する子孫ノードの集合

また，x_i を根とする部分木からの後退辺で関連する x_i の祖先ノードに $P(x_i)$ を加えたノードの数を x_i における擬似的な木の幅とし，$W(x_i)$ により表す．全ての x_i についての $W(x_i)$ の最大値を擬似的な木の全体についての幅とする．

5.3.2 擬似的な木に対する動的計画法の適用

一般的な組み合わせ最適化問題では，動的計画に必要な記憶のサイズは指数関数的に増大する場合があるため，記憶のサイズの制限は本質的に必要である．一方で，バックトラックの回数を抑制するために動的計画法を適用することは有効である．特に制約密度が小さく擬似的な木の幅が小さい問題は，動的計画法のみによる解法を適用できる．以下では動的計画法による基本的な解法を示す．この解法では前処理により，(1) 擬似的な木が既に生成されていて，(2) 上位ノードの値域を下位の近傍ノードが予め知る，ものとする．

特殊な場合として，制約網が木であるときは，後退辺が存在しない擬似的な木を生成できる．このような問題について，木の深さに対する多項式時間の複雑度，値域に対する多項式の空間複雑度の動的計画法による解法を適用できる [42][45]．これは基本的には次のような 2 段階の処理により構成される．

(1) コストの計算

各ノード x_i は，親ノード $P(x_i)$ の全ての変数値 $d_{P(x_i)} \in D_{P(x_i)}$ について，x_i を根とする部分木のコストのベクトル $Util_{x_i,P(x_i)}$ を次式により計算する．この式は木に従ってボトムアップに計算される．

$$Util_{x_i,P(x_i)}(d_{P(x_i)}) = \min_{d_i \in D_i} \{ f_{i,P(x_i)}(d_i, d_{P(x_i)}) + \sum_{x_k \in C(x_i)} Util_{x_k,x_i}(d_i) \} \tag{5.1}$$

(2) 最適解の決定

コストの計算の後， 各ノード x_i は，最適な変数値 d_i^* を次式により決定する．こ

の式は木に従ってトップダウンに計算される．

$$d_i^* = \mathrm{argmin}_{d_i \in D_i}\{f_{i,P(x_i)}(d_i, d^*_{P(x_i)}) + \sum_{x_k \in C(x_i)} Util_{x_k,x_i}(d_i)\} \tag{5.2}$$

一方，一般的な制約網に対する擬似的な木には後退辺が存在する．このため，動的計画法を適用する場合には各ノードは親ノードに加えて，自ノードを根とする部分木から後退辺で関連する全ての祖先ノードについて，値域の組に対するコストの計算が必要になる．このため x_i で計算されるコスト $Util_{x_i,P(x_i)}$ は hyper-cube になる．擬似的な木について一般化された動的計画法に基づく解法である DPOP [46] の手続きを図 5.2 に示す．[1]

この解法は，後退辺が無い場合と同様に，木の深さの 2 倍のメッセージサイクルと辺の数の 2 倍のメッセージ数を要する．一方で，擬似的な木の幅と変数の値域が増加するに従い，各ノードの記憶とメッセージのサイズについて空間複雑度が指数関数的に増大する．従って，問題および，木の構築の際に，擬似的な木の幅を制限することが必要になる．このために制約密度が小さい問題に制限することや木の生成の際に適切な手法を用いることが考えられるが，一般的な問題では擬似的な木の幅を幅を一定値以下にすることは厳密には難しい場合があると考えられる．

[1] 文献 [46] では利得をを最大化する問題として形式化されているが，本論文ではコストを最小化する問題として形式化した．また，便宜上アルゴリズムの表記を変更した．

[**Initialize**]
  $agentview_i \leftarrow \phi$;
  if x_i is a leaf node **then**
    $Util_{x_i,P(x_i)} \leftarrow$ **compute_Util()**;
    send (**UTIL**, x_i, $Util_{x_i,P(x_i)}$) to $P(x_i)$; **endif**
[**UTIL_message_handler**(**UTIL**, x_k, $Util_{x_k,x_i}$)]
  store $Util_{x_k,x_i}$;
  if UTIL messages from all children was received
  then
    if x_i is the root node **then**
      $d_i^* \leftarrow$ **choose_optimal()**;
      send (**VALUE**, x_i, $\{(x_i, d_i^*)\}$) to all $C(x_i)$;
    else $Util_{x_i,P(x_i)} \leftarrow$ **compute_Util()**;
      send (**UTIL**, x_i, $Util_{x_i,P(x_i)}$) to $P(x_i)$;
    endif endif
[**VALUE_message_handler**(**VALUE**, x_j, $agentview_j$)]
  $agentview_i \leftarrow agentview_j$;
  $d_i^* \leftarrow$ **choose_optimal()**;
  send (**VALUE**, x_i, $agentview_i \cup \{(x_i, d_i^*)\}$)
    to all $C(x_i)$;
[**compute_Util()**]
  $\overline{X} = P(x_i) \cup PP(x_i) \cup$
  {ancestor nodes related by $\forall x_k \in C(x_i), Util_{x_k,x_i}$},
  $\forall \overline{d} \in \prod_{x_j \in \overline{X}} D_{x_j}$,
  $Util_{x_i,P(x_i)}(\overline{d}) \leftarrow \min_{d_i \in D_i}\{f_{i,P(x_i)}(d_i, d_{P(x_i)}) +$
    $\sum_{x_j \in PP(x_i)} f_{i,j}(d_i, d_{x_j}) +$
    $\sum_{x_k \in C(x_i)} Util_{x_k,x_i}((\overline{d}, d_i))\}$;
[**choose_optimal()**]
  $d_i^* \leftarrow \mathrm{argmin}_{d_i \in D_i}\{f_{i,P(x_i)}(d_i, d^*_{P(x_i)}) +$
    $\sum_{x_j \in PP(x_i)} f_{i,j}(d_i, d^*_{x_j}) +$
    $\sum_{x_k \in C(x_i)} Util_{x_k,x_i}((agentview_i, d_i))\}$;

図 5.2: DPOP アルゴリズム

5.4　提案手法 — 擬似的な木の幅を考慮した記憶のサイズの制限

5.4.1　擬似的な木の幅を考慮した記憶の制限とバックトラッキングの導入

記憶のサイズを削減する方法として，一部の制約辺について上位の変数値を固定して探索空間を削減することが考えられる．しかし，網羅的な最適解の探索のためには変数値の変更が必要であるため，コストの集計後に値を変更する．すなわち，一部の変数についてバックトラックを導入する．例えば図 5.3(a) では x_i で幅 $W(x_i)$ が 4 であるが，変数 x_m を定数値とすることで，**UTIL** メッセージの hyper-cube のサイズは幅 $W(x_i)$ が 3 の場合と等しくなる．一方で x_m の値の探索のためにバックトラックが必要となる，そこで図 5.3(b) のように，現在の x_m の値についての **UTIL** メッセージによるコストの集計ごとに，**BT** メッセージによって x_m の値の変更を通知する．

バックトラックの対象とする変数は前処理によって選択する．擬似的な木の幅についての制限値 $\overline{W}$ をパラメータとし，$W(x_i) > \overline{W}$ となる場合にその原因となる一部の上位ノードの変数をバックトラックの対象とする．このようなバックトラックの対象とする変数は一意には決定はできない．本論文では，より上位の変数をバックトラックの対象として優先的に選択する．これはバックトラックの対象とする変数の個数を削減し，広範囲の木の幅を削減することを意図する．前処理の概要を図 5.4 に示す．この処理はボトムアップに実行される．結果として，f_i^{bt}

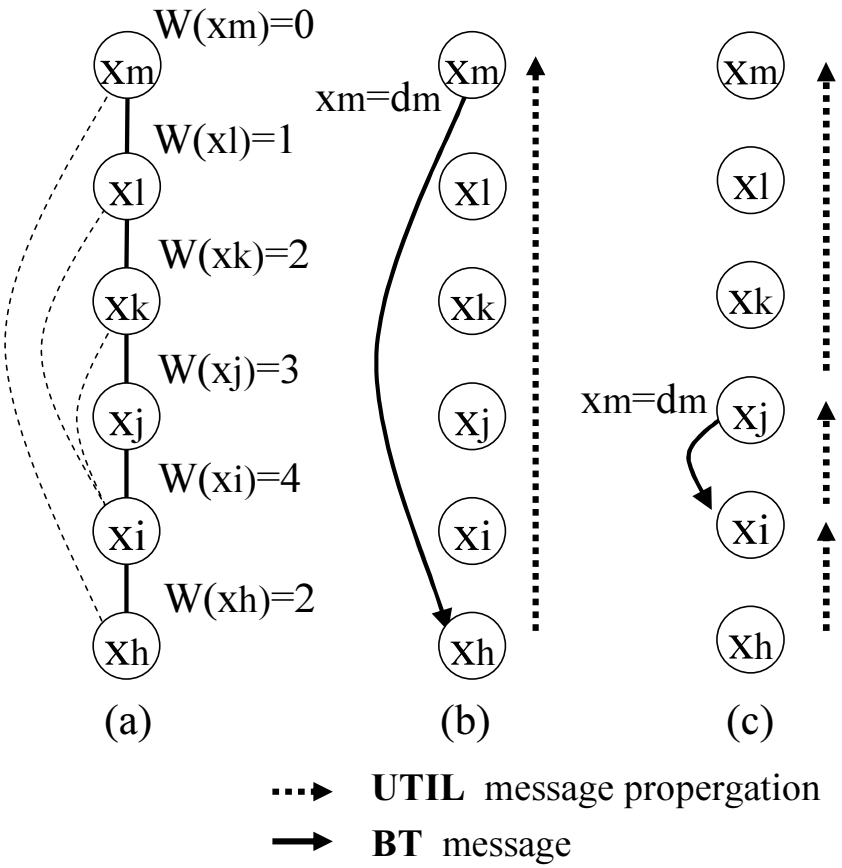


図 5.3: 部分的なバックトラッキング

[**Intialize**]
　$X_i^{all} \leftarrow P(x_i) \cup PP(x_i)$, $X_i^{bt} \leftarrow \phi$, $f_i^{bt} \leftarrow$ false;
　if x_i is a leaf node **then**
　　send (**PRE**, x_i, X_i^{all}, X_i^{bt}) to $P(x_i)$; **endif**
[**PRE_message_Handler**(**PRE**, x_j, X_j^{all}, X_j^{bt})]
　$X_i^{all} \leftarrow X_i^{all} \cup X_j^{all}$, $X_i^{bt} \leftarrow X_i^{bt} \cup X_j^{bt}$;
　if PRE message from all children was received
　then
　　remove x_i from X_i^{all};
　　if x_i in X_i^{bt} **then**
　　　remove x_i from X_i^{bt}; $f_i^{bt} \leftarrow$ true; **endif**
　　while $|X_i^{all} - X_i^{bt}| > \overline{W}$ **do**　　(A)
　　　$X_i^{bt} \leftarrow \{x_k | x_k \in X_i^{all}\, x_k \notin X_i^{bt}\}$; **enddo**
　　send (**PRE**, x_i, X_i^{all}, X_i^{bt}) to $P(x_i)$;
　endif

図 5.4: 前処理 — バックトラックの対象とする変数の決定

により x_i がバックトラックの対象であることを示す．また，この処理では各ノード x_i について，x_i と制約で関連する子孫ノード $C(x_i) \cup PC(x_i)$ のうち，各部分木の最下位に位置するノードからなる集合 $PC^{lowest}(x_i)$ を同時に求める．

バックトラックを導入した DPOP アルゴリズムを図 5.5 に示す．この手法では，変数 x_i がバックトラックの対象であるときノード i は現在の変数値 d_i を **BT** メッセージにより $PC^{lowest}(x_i)$ に含まれるノードに通知する．また，バックトラック対象でない変数値は $*$ により表し，初回のみ **BT** メッセージにより通知する．

ノード x_i が **BT** メッセージを受信したとき，$agentview_i$ に現在の上位ノードの変数値を記録する．また，**BT** メッセージで直接通知されない変数値は，**UTIL** メッセージにより子孫ノードから伝搬される．これにより $agentview_i$ は現在の祖先ノードの変数値が記録される．$agentview_i$ に含まれる変数値が $*$ では無い変数

[**Initialize**]
  $agentview_i \leftarrow \phi$; $value\_was\_received \leftarrow$ false;
  if f_i^{bt} **then** $d_i \leftarrow d \in D_i$; **else** $d_i \leftarrow *$; **endif**
  send (**BT**, x_i, d_i) to all $PC^{leaf}(x_i)$;
[**BT_message_handler**(**BT**, x_j, d_j)]
  if not $value\_was\_received$ **then**
    add (x_j, d_j) to $agentview_i$; **endif**
  maintain();
[**UTIL_message_handler**(**UTIL**, x_k, $agentview_k$, $Util_{x_k,x_i}$)]
  if not $value\_was\_received$ **then**
    replace $agentview_i$ by $agentview_k$; **endif**
  store ($agentview_k$, $Util_{x_k,x_i}$); **maintain()**;
[**VALUE_message_handler**(**VALUE**, x_j, $agentview_j$)]
  $agentview_i \leftarrow agentview_j$;
  $value\_was\_received \leftarrow$ true; **maintain()**;
[**maintain()**]
  forall stored ($agentview_k$, $Util_{x_k,x_i}$) **do**
  if $agentview_k$ is incompatible with $agentview_i$
    then remove ($agentview_k$, $Util_{x_k,x_i}$);
      endif enddo
   if UTIL messages from all children was received
   then
    if f_i^{bt} **then**
      compute partial $Util_{x_i,P(x_i)}$ for d_i; (A)
      if $\exists d \in D_i$ is not searched **then**
        $d_i \leftarrow$ such d;
        send (**BT**, x_i, d_i) to all $PC^{leaf}(x_i)$;
      endif endif
    if $value\_was\_received$ **or** x_i is the root node
    then $d_i^* \leftarrow$ **choose_optimal()**;
      if f_i^{bt} **then**
        send (**BT**, x_i, d_i^*) to all $PC^{leaf}(x_i)$; **endif**
      send (**VALUE**, x_i, $agentview_i \cup \{(x_i, d_i^*)\}$)
        to all $C(x_i)$;
    else $Util_{x_i,P(x_i)} \leftarrow$ **compute_Util()**; (B)
      send (**UTIL**, x_i, $Util_{x_i,P(x_i)}$) to $P(x_i)$;
     endif endif

図 5.5: DPOP とバックトラックの併用 (DPOP+BT1)

については，変数値を固定し，$*$ である変数については従来手法と同様に各値域の組のコストの計算を行う．

これまでに子ノード k から受信した $(agentview_k, Util_{x_k,x_i})$ について，$agentview_k$ が $agentview_i$ と矛盾するときは，その $(agentview_k, Util_{x_k,x_i})$ は削除される．ここで，$agentview_k$ が $agentview_i$ について無矛盾であるとき，

$$\forall x_j, \quad (x_j, d_j^k) \in agentview_k, (x_j, d_j^i) \in agentview_i,$$
$$d_j^k = * \vee d_j^k = d_j^i$$

である．このような $agentview_i$ の使用法はバックトラックを用いる分散最適化手法である ADOPT[16] で用いられる current context を変数値が $*$ で表される場合について一般化したものである．

コストの計算 **compute_Util()** および最適解の決定 **choose_optimal()** は一部の変数値が固定されること除いて DPOP と同様である．たとえば，変数 x_m がバックトラックの対象であり，現在の値が d_m であるとき，x_m に関連する下位のノード x_i における $Util_{x_i,P(x_i)}$ は，変数 x_m について値域が $D_m = \{d_m\}$ である場合と同様になる．

また，バックトラックの対象である変数 x_m のノードでは，**UTIL** メッセージによるコストの集計は現在の変数値 $d_m \in D_m$ ごとに行われる．このため，図 5.5(A) の箇所で各 d_m についての $Util_{x_m,P(x_m)}$ を求めておき，最終的に図 5.5(B) の箇所で $\forall d \in D_m$ について統合された $Util_{x_m,P(x_m)}$ が得られているものとした．また，再計算を削減するために，$Util_{x_m,P(x_m)}$ の各要素に対応する変数値 d_m^* も記録しておくものとした．

5.4.2 バックトラックの範囲の削減

バックトラックの処理は擬似的な木の幅が制限値を超える部分について適用することが適切である．しかし，単純にあるノードで変数を変更する方法ではバックトラックの際に再計算が行われる範囲が冗長な広範囲に及ぶ可能性がある．例えば，根ノードと，最も深い葉ノードとの間に制約辺がある場合に，根ノードの変数を変更すると，これらの 2 ノード間でコストの再計算が必要になる．一方，擬似的な木の幅が制限値を越える場所はそれらの中間の部分のみの場合がある．

このような冗長性を削減し，擬似的な木の幅が制限値を超えた部分のみバックトラックを適用するために以下の 2 つの方法を用いる．

(1) 擬似的な木の幅 $W(x_h)$ が制限値を超えないノード x_h では，上位の変数 x_m がバックトラックの対象となる場合でも x_m の全ての値域についてのコストの集計を行う．また，その上位で擬似的な木の幅 $W(x_i)$ が制限値を超えたノード x_i について $x_i \in PC^{leaf}(x_i)$ とし，x_i を x_m からの **BT** メッセージを受信するノードとする．すなわちバックトラックを行う区間の下側を擬似的な木の幅が制限値を超えるノードまで上位に移動する．

(2) 擬似的な木の幅 $W(x_j)$ が制限値を超えないノード x_j では，ノード x_j を根とするサブツリーに，x_j の上位ノード x_m の **BT** メッセージを受信するノードがある場合，x_j は x_m の代わりにバックトラック処理を行う．すなわちバックトラックを行う区間の上側を擬似的な木の幅が制限値を超える直前ノードまで下位に移動する．

これらにより，図 5.3(c) のように，木の幅が制限を超える部分について，バックトラックが行われる．このために 5.4.1 の手法を拡張する．前処理では，図 5.4(A)

の $|X_i^{all} - X_i^{bt}| > \overline{W}$ の場合に X_i^{bt} に変数名を追加する処理に加えて，変数を除去する処理を行う．すなわち，$|X_i^{all} - X_i^{bt}| < \overline{W}$ の場合は，X_i^{bt} から変数名を除去する．また，除去した変数については，ノード i でバックトラック処理を行う．すなわち図 5.5 におけるバックトラックの対象の変数についての処理と同様に，**BT** メッセージの送信および，各変数値ごとの **UTIL** メッセージによるコストを統合する処理をノード i で行う．

5.4.3　近似解法としての利用

上記の手法は単純なバックトラックに基づく網羅的な探索となるため，バックトラックの対象となる変数の増加によりメッセージサイクル数が指数関数的に増加する．このため，実際的には，ある程度の誤差を許容し速やかに解を得ることが必要になる．上記の手法は，バックトラックの対象である変数値の変更回数を制限することで，近似解法とすることができる．本論文では特に簡単な場合として，5.4.1 の手法でバックトラックの対象の対象として選択された変数の値を初期値のままで固定する貪欲戦略について検討する，

このような初期値の選択についての基準は問題に依存すると考えられる．本論文では，擬似的な木に配置された各変数 x_i の変数値 d_i についてのコストの下限値を求める前処理 [18] により得られるコスト値

$$\begin{aligned} h(d_i) \;=\; & \sum_{x_k \in C(x_i)} \{ \min_{d_k \in D_k} \{ h(d_k) + f_{k,i}(d_k, d_i)) + \\ & \sum_{x_j \in PC(x_k)} \min_{d_j \in D_j} f_{k,j}(d_k, d_j) \} \} \end{aligned} \tag{5.3}$$

表 5.1: 擬似的な木の深さの平均

c \ n	25	50	100	200
n-1	6	7	9	11
1.125n	9	15	23	44
1.25n	11	18	32	59
1.5n	12	22	43	77

表 5.2: 擬似的な木の幅の平均

c \ n	25	50	100	200
n-1	1	1	1	1
1.125n	3	4	6	11
1.25n	4	6	10	20
1.5n	5	10	18	33

表 5.3: 平均サイクル数 (DPOP)

c \ n	25	50	100	200
n-1	11	14	17	20
1.125n	17	29	46	–
1.25n	21	36	–	–
1.5n	23	44	–	–

を用い，$h(d_i)$ が最小となる変数値を選択した．

表 5.4: **UTIL** メッセージのサイズの総和の平均 (DPOP)

c \ n	25	50	100	200
n-1	72	147	297	597
1.125n	157	607	6020	–
1.25n	369	5147	–	–
1.5n	1458	235975	–	–

5.5　評価

提案手法についてシミュレーションにより評価した．文献 [46] では比較的制約密度が小さいと考えられるイベントスケジュールなどの問題を例題として用いているが．本論文では問題の規模の議論を簡単にするために，ランダムに生成された 2 項制約で関連する問題を例題として用いた．値域は各ノード i とも $|D_i| = 3$ とした．制約辺に対応付けられた関数の評価値は，制約辺で関連する 2 つの変数の値の各組み合わせについて，1 から 10 の整数値のいずれかとした．

制約網は単一の連結成分を持つように構成し，変数の数 n に対して制約辺の数 c が $n-1$, $1.125n$, $1.25n$, $1.5n$ とし，各 n, c について 10 問の例題を生成した．

深さ優先探索木を生成する際は最大次数のノードを優先して探索するものとした．

シミュレーションでは，システム全体はメッセージサイクルを単位とする反復処理を行い，各メッセージサイクルでは次の (1)(2) の処理を行うものとした．(1) 各ノードはそれぞれ受信メッセージキューと送信メッセージを持つ．各ノードは受信メッセージキューにあるメッセージを読み出し，処理の結果に基づいてメッセージを送信メッセージキューに書き込む．(2) 各ノードの送信メッセージキュー

表 5.5: DPOP とバックトラックの併用 (木の幅の制限=4)

n	25	50	100
c	1.5n	1.25n	1.125n
num. of cycles			
DPOP	23	36	46
DPOP+BT1	102	344	322
DPOP+BT2	41	115	126
total size of UTIL messages			
DPOP	1458	5147	6020
DPOP+BT1	1854	7534	7134
DPOP+BT2	1612	5884	6757

表 5.6: DPOP とバックトラックの併用 (木の幅の制限=8)

n	50	100	200
c	1.5	1.25	1.125
num. of cycles			
DPOP	44	–	–
DPOP+BT1	249	1188	4295
DPOP+BT2	84	217	891
total size of UTIL messages			
DPOP	235975	–	–
DPOP+BT1	271246	963243	4713141
DPOP+BT2	250409	823191	4480157

にあるメッセージを，あて先ノードの受信メッセージキューに移動する．

5.5.1 生成された擬似的な木と DPOP の評価

例題の制約網に対して生成された擬似的な木の深さを表 5.1 に，擬似的な木の幅を表 5.2 に示す．

これらの問題について，DPOP に基づく手法を実行した際に要したサイクル数を表 5.3 に，**UITL** メッセージで送信されたコストのサイズの総和を表 5.4 に示す．これらの結果では，**UTIL** メッセージに含まれるコストのサイズの最大値が 10^6 を超える問題が含まれる場合については空欄とした．表 5.3 に示されるように，DPOP の実行に要するサイクル数は表 5.1 に示される木の深さの 2 倍の程度となる．

5.5.2　バックトラックによる遅延の評価

バックトラックを部分的に導入する場合，明らかにバックトラックの対象となる変数の個数に従って指数関数的にメッセージサイクル数が増大する．このため実際には 5.4.3 で述べたような手法の併用が必要である．ここでは，特にバックトラック処理による遅延について評価することを目的とし，高々数個の変数をバックトラックの対象とする場合について評価した．

擬似的な木の幅の制限値を統一するために，表 5.2 の (n,c) の組について，木の幅の平均が比較的近い $\{(25,\ 1.5n),\ (50,\ 1.25n),\ (100,\ 1.125n)\}$, $\{(50,\ 1.5n),\ (100,\ 1.25n),\ (200,\ 1.125n)\}$ について，それぞれ木の幅の制限値を 4, 8 とした場合について評価した．DPOP と 5.4.1 の手法 (DPOP+BT1) および 5.4.2 の手法 (DPOP+BT2) について比較した．

表 5.5 に擬似的な木の幅の制限値が 4 の場合，表 5.6 に擬似的な木の幅の制限値が 8 の場合の結果をそれぞれ示す．いずれの場合も，メッセージサイクル数については DPOP よりもバックトラックを行う DPOP+BT1,DPOP+BT2 の方が大きい．バックトラックの範囲を削減する DPOP+BT2 は DPOP+BT1 よりもメッセージサイクル数は少ない．また，**UTIL** メッセージのサイズの総和は，DPOP よ

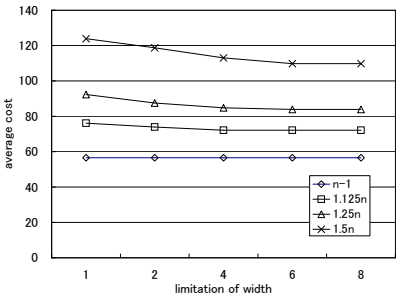
りも DPOP+BT1,DPOP+BT2 の方が大きい．これは，DPOP+BT1,DPOP+BT2 では，各ノード i は現在の $agentview_i$ についての記録されたコストのみを記録しているため，上位ノードの選択した最適解が $agentview_i$ が矛盾する場合に一部のコストを再計算する必要があることによる．

5.5.3 変数値を固定した場合の評価

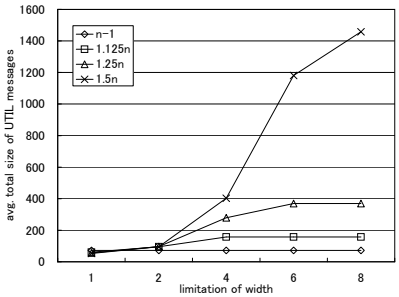
バックトラックの対象とする変数値を固定した場合の評価として，各問題について擬似的な木の幅の制限値を 1, 2, 4, 6, 8 としたときのコストの平均と **UTIL** メッセージのサイズの総和の平均を図 5.6 に示す．制限値を大きくすることによりコストは改善するが，**UTIL** メッセージのサイズの総和については，擬似的な木の幅が大きい場合は制限値を大きくするに従って従い指数関数的に増大する．制約密度の低い問題では，一部の変数値を固定しても他の変数値の変更により，コストを改善できる余裕があることや，コストの下限値による貪欲戦略が比較的有効であることが考えられる．このような場合は近似解を採用することが実際的であると考えられる．

5.6 関連研究と考察

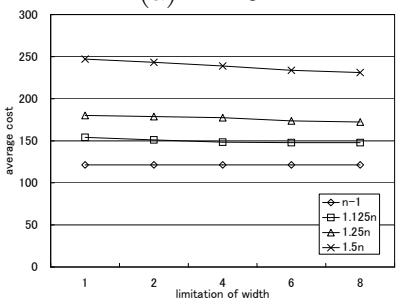
提案手法は，動的計画法に特化した手法である DPOP[46] の一部の処理をバックトラックに置き換えるものである，この基本的なアイデアは [46] でも指摘されているが，本稿では，バックトラックに基づく解法である ADOPT[16] の機構の一部を導入し，ADOPT と DPOP の中間的な動作を行うための手法を示した．ま



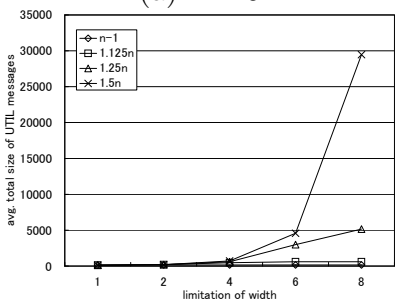
(a) n=25



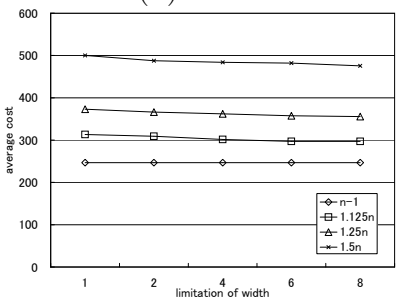
(a) n=25



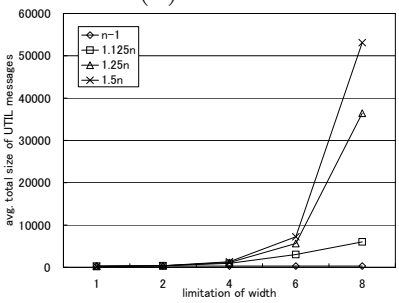
(b) n=50



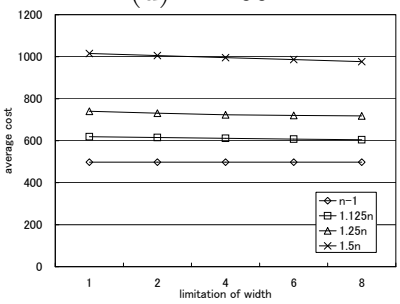
(b) n=50



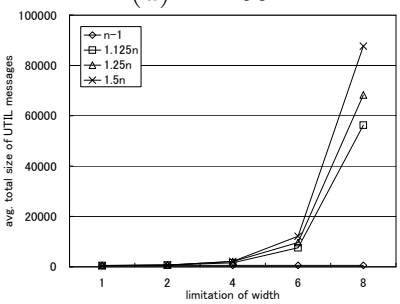
(d) n=100



(d) n=100



(d) n=200
(コストの平均)



(d) n=200
(**UTIL** メッセージのサイズの総和)

図 5.6: 変数値を固定した場合

た，さらにADOPT同様に各ノードでコストの上下界にもとづきバックトラックを行うように一般化することも可能であると考えられる．しかし，コストの上下界に基づく効率的なバックトラックの導入のためには，各**UTIL**メッセージの処理により探索される解の範囲についての検討が必要であると考えられる．

記憶のサイズの制限のために**UTIL**メッセージで送受されるコストの表現を変更し冗長性を除くことが考えられる．3章で示した複数の解とコストを内包関係を利用し統合する手法をメッセージの表現に応用することが考えられる．また，記憶のサイズを制限するために，コスト評価についての情報の一部を省略するような近似的な解法を用いる場合は，記憶のサイズと解の精度のトレードオフについての検討が必要である．

5.7 まとめ

本章では，擬似的な木と動的計画法に基づくこの分散制約最適化手法の補助的な手段として，擬似的な木の幅が制限される値を超える部分について記憶のサイズを制限するための基本的な手法を提案した．一部の変数について部分的にバックトラックを導入する手法，および，貪欲戦略の一つとして一部の変数の変数値を予め固定する手法を示した．また，提案手法の有効性について評価した．

一般的な組み合わせ最適化問題では動的計画法は指数関数的な空間複雑度を必要とする．従って，問題を構成する際に制約密度に制限を設けることや，擬似的な木の幅を小さくするようなノードの順序付けを生成することが重要である．提案手法はこのような制限の補助的な手段として利用することが適切であると考えら

れる．また，5.6 で述べたように Adopt と DPOP の中間的な手法については，コストの上下界や部分解の内包関係を考慮した，より効率的な手法の検討が必要である．上記を含めた，実際的な問題への適用が今後の課題である．

第6章　結論

以下ではでは本論文の内容についてについてまとめ，今後の課題について述べる．

第 1 章では，本研究の背景および目的について述べた．この中で，分散制約最適化問題の重要性および本研究において特に議論の対象とする解法，およびその研究課題を挙げた．また，本論文の構成を示した．

第 2 章では分散制約最適化問題についての概論を示した．まず，分散制約最適化問題の基礎となる，分散制約充足問題に関する研究について述べた．次に，より一般的な組み合わせ最適化問題として形式化された，分散制約最適化問題について述べた．分散制約充足問題から分散制約最適化問題への問題の拡張および，各問題の解法についての関連研究を概観した．

第 3 章ではメモリ制限型の A*アルゴリズムに基づく非同期分散探索手法 [15] をもとに，探索処理の効率化手法について述べた．まず，対象とする問題の形式化および従来手法の概要を示し，提案手法として 2 つの効率改善手法を示した．1 つめの効率改善手法として，バックトラッキングの効率化のために，コストの下界値に基づく部分解の導出と逐次処理のバックトラック型の探索における Backjumping に類する手法を導入した．2 つめの効率改善手法として，メモリ制限型の A*アルゴリズムについて，メモリの制限を緩和し，複数の部分解とコストを学習する手法を導入した．提案手法の有効性を計算機実験により示した．

第 4 章では第 3 章で検討した解法について，動的な環境への適用について述べた．対象とする動的な問題を形式化し，提案手法を示した．本研究では特に，制約網の変化の観測，制約網の再構築，制約網に対する木の際構築, 探索の実行と解の決定の一連の処理をボトムアップな非同期分散処理として統合する．このような一連の処理を反復的に動作させることで，動的に変化する制約網に対し，追従する枠組みを示した．

第 5 章では，関連研究に関する検討として，擬似的な木を用いる動的計画法に特化した解法として提案された分散制約最適化手法 [46] について，メモリ制限のため基本的な手法を検討した．深さ優先探索木などの擬似的な木による変数の順序に基づく探索手法に動的計画法を適用する場合，メッセージ数とメッセージサイクル数は木の深さに対する多項式時間の複雑度となるが，空間複雑度は擬似的な木の幅に対して指数関数的に増大する．この擬似的な木の幅を考慮し，木の幅が大きい箇所について，変数にバックトラックを導入する手法を示した．

本論文においては，特に，制約網に対する深さ優先探索木とメモリ制限型の A* アルゴリズムにもとづく制約最適化手法 [16] を議論の対象とし，第 3 章のの探索の効率改善手法の導入，および，第 4 章の動的な問題への適用についての検討が研究の主体となっている．これらの内容は DCOP の解法をもとにした分散システムの実装のための基礎的な検討として位置づけられる．提案手法を応用した，ネットワーク上の資源割り当て処理などの，実際的なシステムの構築と検証が今後の課題である．

DCOP に関する最近の動向として，本論文で特に議論の対象とした，制約網に対する擬似的な木に基づく手法 [16],[46] 以外にも，分散制約最適化問題に対する

完全解法 [47] や近似的な解法が提案されている．また応用的な問題として，センサ網における観測資源の割り当て問題，イベントスケジューリングへの適用などが提案されている [17]．動的な問題については [46] をもとにした自己安定アルゴリズムも提案されており，これらの手法についての検討が今後の課題である．

謝辞

本研究を行うにあたり，終始にわたりご指導とご助言を賜りました名古屋工業大学 大学院 工学研究科 情報工学専攻 松尾啓志 教授，名古屋工業大学 大学院 工学研究科 情報工学専攻 岩田彰 教授に深く感謝申し上げます.

本論文の作成にあたり有益なご指導を賜りました九州大学大学院システム情報科学研究院 知能システム学部門 横尾真教授, 名古屋工業大学 大学院 工学研究科 情報工学専攻 犬塚信博助教授に深く感謝申しあげます.

名古屋工業大学大学院 工学研究科 情報工学専攻 黒柳奨助手，若山公威助手，岩田研究室ならびに松尾研究室の皆様には大変お世話になりました．皆様のご支援に感謝申し上げます.

参考文献

[1] G. Weiss, editor, "Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence," The MIT Press, 2000.

[2] A. K. Mackworth, "Constraint satisfaction," S. C. Shapiro, editor, Encyclopedia of Artificial Intelligence, pp.285–293. John Wiley & Sons, 1992.

[3] 西原清一, "制約充足問題の基礎と展望," 人工知能学会誌, vol.12, no.3, pp.351-358, 1997.

[4] 横尾真, 平山勝敏, "CSP の新しい展開:分散/動的/不完全 CSP," 人工知能学会誌, vol.12, no.3, pp.381-389, 1997.

[5] E. C. Freuder and R. J. Wallace, "Partial Constraint Satisfaction," Artif. Intell., vol.58, no.1–3, pp.21–70, 1992.

[6] T. Schiex, H. Fargier and G. Verfaillie, "Valued constraint satisfaction problems: Hard and easy problems," Proc. Int. Joint Conf. on Artificial Intelligence, vol.1, pp.631–639, 1995.

[7] G. Verfaillie, M. Lemaître and T. Schiex, "Russian Doll Search for Solving Constraint Optimization Problems," Proc. 13th National Conf. on Artificial Intelligence, vol.1, pp.181–187, 1996.

[8] M. Yokoo, E. H. Durfee, T. Ishida and K. Kuwabara, "Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving," Proc. 12th IEEE Int. Conf. on Distributed Computing Systems, pp.614–621, 1992.

[9] M. Yokoo, E. H. Durfee, T. Ishida and K. Kuwabara, "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms,"IEEE Trans. Knowledge and Data Engineering, vol.10, no.5, pp.673–685, 1998.

[10] M. Lemaître and G. Verfaillie, "An Incomplete Method for Solving Distributed Valued Constraint Satisfaction Problems," Proc. AAAI97 Workshop on Constraints and Agents, 1997.

[11] K. Hirayama and M. Yokoo, "Distributed Partial Constraint Satisfaction Problem," Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming, pp.222–236, 1997.

[12] K. Hirayama and M. Yokoo, "An Approach to Over-constrained Distributed Constraint Satisfaction Problems: Distributed Hierarchical Constraint Satisfaction," Proc. 4th Int. Conf. on Multiagent Systems, pp.135–142, 2000.

[13] J. Liu and K. Sycara, "Exploiting problem structure for distributed constraint optimization," Proc. 1st Int. Conf. on Multiagent Systems, pp.246–253, 1995.

[14] Y. Hamadi, "Interleaved search in distributed constraint networks," International Journal on Artificial Intelligence Tools, vol.3, no.4, pp.167–188, 2002.

[15] P. J. Modi, W. Shen, M. Tambe and M. Yokoo, "An Asynchronous Complete Method for Distributed Constraint Optimization," Proc. Autonomous Agents and Multi-Agent Systems, pp.161–168, 2003.

[16] P. J. Modi, W. Shen, M. Tambe and M. Yokoo, "Adopt: asynchronous distributed constraint optimization with quality guarantees," Artif. Intell., vol.161, no.1–2, pp.149–180, 2005.

[17] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce and P. Varakantham, "Taking DCOP to the Real World : Efficient Complete Solutions for Distributed Event Scheduling," Proc. 3rd Int. Conf. on Agents and Multi Agent Systems, pp.310–317, 2004.

[18] S. Ali, S. Koenig and M. Tambe, "Preprocessing Techniques for Accelerating the DCOP Algorithm ADOPT, "Proc. 4th Int. Conf. on Autonomous Agents and Multiagent Systems, pp.1041–1048, 2005.

[19] D. L. Mammen and V. R. Lesser, "Problem Structure and Subproblem Sharing in Multi-Agent Systems," Proc. 3rd Int. Conf. on Multiagent Systems, pp.174–181, 1998.

[20] K. Hirayama and M. Yokoo, "The Effect of Nogood Learning in Distributed Constraint Satisfaction," Proc. 20th Int. Conf. on Distributed Computing Systems, pp.169–177, 2000.

[21] T. Schiex and G. Verfaillie, "Nogood Recording for Static and Dynamic Constraint Satisfaction Problems," International Journal of Artificial Intelligence Tools, vol.3, no.2, pp.187–207, 1994.

[22] M. Yokoo, "Weak-commitment Search for Solving Constraint Satisfaction Problems," Proc. 12th National Conf. on Artificial Intelligence, pp. 313–318, 1994.

[23] M. Yokoo, "Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems," Proc. 1st Int. Conf. on Principles and Practice of Constraint Programming, pp.88–102, 1995.

[24] P. Morris, "The breakout method for escaping from local minima," In Proc. AAAI-93, pp.40–45, 1993.

[25] M. Yokoo and K. Hirayama, "Distributed breakout algorithm for solving distributed constraint satisfaction problems," orc. 2nd Int. Conf. on Multiagent Systems, pp.401–408, 1996.

[26] M. Yokoo and K. Hirayama, "Distributed Constraint Satisfaction Algorithm for Complex Local Problems," Porc. 3rd Int. Conf. on Multiagent Systems, pp.372–379, 1998.

[27] B. Awerbuch, "A New Distributed Depth-First-Search Algorithm," Inf. Process. Lett., vol.20, no.3, pp.147–150, 1985.

[28] M. B. Sharma and S. S. Iyengar, "An Efficient Distributed Depth-First-Search Algorithm," Inf. Process. Lett., vol.32, no.4, pp.183-186, 1989.

[29] D. Kumar, S. S. Iyengar and M. B. Sharma, "Corrigenda: Corrections to a Distributed Depth-First Search Algorithm," Inf. Process. Lett., vol.35, no.1, pp.55-56, 1990.

[30] R. Dechter and A. Dechter, "Belief Maintenance in Dynamic Constraint Networks", Proc. 7th National Conf. on Artificial Intelligence, pp.37–42, 1988.

[31] G. Verfaillie and T. Schiex, "Solution Reuse in Dynamic Constraint Satisfaction Problems," Proc. 20th National Conf. on Artificial Intelligence, pp.307–312, 1994.

[32] R. J. Wallace and E. C. Freuder, "Stable solutions for dynamic constraint satisfaction problems," Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming pp.447–461, 1998.

[33] 服部 宏充, 磯村 厚誌, 伊藤 孝行, 大囿 忠親, 新谷 虎松, "動的重み付き最大制約充足問題に基づくナーススケジューリングシステム暫定制約の導入に基づく解の安定性の実現," 人工知能学会論文誌, vol.20, no.1, pp.25–35, 2005.

[34] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," ACM Trans. on Computer Systems, vol.3, no.1, pp.63–75, 1985.

[35] K. Venkatesh, T. Radhakrishnan and H.F. Li, "Optimal checkpointing and local recording for domino-free rollback recovery," Inf. Process. Lett., vol. 25, no. 5, pp.295– 304, 1987.

[36] W. Dijkstra and C. S. Scholten, "Termination Detection for Diffusing Computations," Inf. Process. Lett., vol. 11, no.1, pp.1–4, 1980.

[37] N. Shavit and N. Francez, "A new approach to detection of locally indicative stability," International Colloquium on Automata, Languages and Programming on Automata, lanuages and programming, pp.344-358, 1986.

[38] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," Proc. 7th National Conf. on Artificial Intelligence, vol.1, pp.49–54. 1988.

[39] E. C. Freuder and M. J. Quinn, "Taking advantage of stable sets of variables in constraint satisfaction problems," Proc. of the 9th Int. Joint Conf. on Artificial Intelligence, pp.1076–1078, 1985.

[40] E. C. Freuder, "A sufficient condition for backtrack-bounded search," Journal of the ACM, vol.32, no.14, pp.755–761, 1985.

[41] T. Schiex, "A note on CSP graph parameters," Technical Report 03, INRA, 1999.

[42] F. R. Kschischang, B. Frey and H. Andrea Loeliger, "Factor graphs and the sum-product algorithm," IEEE Trans. on Information Theory, vol.47, no.2), pp.498–519, 2001.

[43] R. Marinescu and R. Dechter, "AND/OR Tree Search for Constraint Optimization", 6th Int. Workshop on Preferences and Soft Constraints of the 10th Int. Conf. on Principles and Practice of Constraint Programming, 2004.

[44] R. Marinescu and R. Dechter, "Advances in AND/OR Branch-and-Bound Search for Constraint Optimization," 7th Int. Workshop on Preferences and Soft Constraints of the 11th Int. Conf. on Principles and Practice of Constraint Programming, 2005.

[45] A. Petcu and B. Faltings, "A distributed, complete method for multi-agent constraint optimization," 5th Int. Workshop on Distributed Constraint Reasoning, 2004.

[46] A. Petcu and B. Faltings, "A Scalable Method for Multiagent Constraint Optimization," Proc. 9th Int. Joint Conf. on Artificial Intelligence, pp.266–271, 2005.

[47] R. T. Mailler and V. Lesser, "Solving Distributed Constraint Optimization Problems Using Cooperative Mediation," Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and MultiAgent Systems, pp.438–445, 2004.

[48] R. Bejar, C. Domshlak, C. Fernandez, C. Gomes, B. Krishnamachari, B. Selman and M. Valls, "Sensor networks and distributed CSP: communication, computation and complexity," Artif. Intell, vol.161 no.1–2, pp.117-147, 2005.

本論文に関する研究業績等

学術論文

[1] **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,” 電子情報通信学会論文誌, vol.J88-D-I, no.8, pp.1235–1246, 2005 年 8 月.

[2] **松井 俊浩**, 松尾 啓志, “深さ優先探索木に基づく分散制約最適化手法の動的問題への適用のための一手法,” 電子情報通信学会論文誌,D.(投稿中)

[3] **松井 俊浩**, 松尾 啓志, “擬似的な木を用いた分散制約最適化手法のための木の幅を考慮した空間複雑度の削減,” 電子情報通信学会論文誌,D.(投稿中)

国際会議

[1] **Toshihiro Matsui** and Hiroshi Matsuo, “Applying Distributed Constraint Optimization Method to Dynamic Problem,” *Proc. of Computational Intelligence (CI2005)*, pp.170–175, Jul. 2005.

[2] **Toshihiro Matsui**, Hiroshi Matsuo and Akira Iwata, “Efficient Method for Asynchronous Distributed Constant Optimization Algorithm,” *Proc. of*

Artificial Intelligence and Applications (AIA2005), pp.727–732, Feb. 2005.

[3] **Toshihiro Matsui**, Hiroshi Matsuo and Akira Iwata, "Dynamic camera allocation method based on constraint satisfaction and cooperative search," *Proc. of 2nd International Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD2001)*, pp.955–964, Aug. 2001.

技術報告

[1] **松井 俊浩**, 松尾 啓志, 岩田 彰，"分散制約充足問題の観測資源割り当てへの適用," 情報処理学会研究会報告コンピュータビジョンとイメージメディア, 2002-CVIM-131，pp.89-95, 2002 年 1 月.

国内学会発表

[1] **松井 俊浩**, 松尾 啓志，"擬似的な深さ優先探索木を用いた動的な分散制約最適化のための枠組み," 平成 17 年度電気関係学会東海支部連合大会，2005 年 9 月.

[2] **松井 俊浩**, 松尾 啓志，"動的な分散制約最適化問題のための基本的な枠組みの提案," 人工知能学会第 19 回全国大会 (JSAI2005)，2005 年 6 月.

[3] **松井 俊浩**, 松尾啓志，岩田 彰，"動的分散制約最適化問題への深さ優先探索木に基づく解法の適用," 情報処理学会第 67 回全国大会，2005 年 3 月.

[4] **松井 俊浩**, 松尾 啓志, 岩田彰, “分散制約最適化問題のための非同期分散探索における上下界を用いた効率改善手法,” 平成 16 年度電気関係学会東海支部連合大会, 2004 年 9 月.

図 目 次

3.1 Adopt のメッセージ 24
3.2 サイクル数 (制約重み=1,d=2) 45
3.3 サイクル数 (制約重み=1,d=3) 45
3.4 サイクルあたりのメッセージ数 (制約重み=1,d=3) 49
3.5 サイクル数 (制約重み=$\{1,\cdots,10\}$,d=2) 49
3.6 サイクル数 (制約重み=$\{1,\cdots,10\}$,d=3) 50
3.7 キャッシュ長のサイクル数への影響 (sc+lru+integrate, 制約重み=1,d=3) 51
3.8 変数値配信パス変更の影響 (制約重み=1,d=2,3) 51
3.9 MPI環境における最大サイクル数の平均 (制約重み=$\{1,\cdots,10\}$,d=3,1台) 54

4.1 処理の概要 61
4.2 処理の枠組み 66
4.3 **STS** メッセージの送受信 66
4.4 木の生成 69
4.5 Adopt のメッセージ 69
4.6 解の決定の同期 73

4.7 解のコストの例 (n=20,d=2,T=1000), 各問題 $p0$〜$p5$ の◆において最適解であることを検出した. 73
4.8 解のコストの例 (n=20,d=3,T=1000) 74
4.9 $Tree_{i,clk_i}$ についての木の構築処理 84
4.10 $Adpt_i^{cur}$ についての終了処理 86
4.11 $Adpt_i^{sub}$ についての処理 86

5.1 制約網と pseudo tree 92
5.2 DPOP アルゴリズム 95
5.3 部分的なバックトラッキング 97
5.4 前処理 — バックトラックの対象とする変数の決定 98
5.5 DPOP とバックトラックの併用 (DPOP+BT1) 99
5.6 変数値を固定した場合 108

表 目 次

3.1 各問題のサイクル数の比較 (制約重み=1,d=2, 括弧内の手法は比較対象) 46
3.2 各問題のサイクル数の比較 (制約重み=1,d=3, 括弧内の手法は比較対象) 47
3.3 各結果の分散 (ノード数 16) 50
3.4 MPI 環境における実行時間 [ms](制約重み=$\{1,\cdots,10\}$,d=3) 53
3.5 各結果の分散 (MPI 環境, 制約重み=$\{1\cdots10\}$,d=3, ノード数 16) . . 54

4.1 最適解の数 ($d=2$, 各 600 問における割合 (%)) 74
4.2 最適解の数 ($d=3$, 各 600 問における割合 (%)) 75
4.3 解のコストの最適解との誤差の平均 75
4.4 問題毎の STS,TREE メッセージ数 78
4.5 問題毎の制約網, 木の構築の収束までのサイクル数 78
4.6 生成された木の深さ 81
4.7 問題あたりのメッセージ数 81

5.1 擬似的な木の深さの平均 103
5.2 擬似的な木の幅の平均 103

5.3 平均サイクル数 (DPOP) 103
5.4 **UTIL** メッセージのサイズの総和の平均 (DPOP) 104
5.5 DPOP とバックトラックの併用 (木の幅の制限=4) 105
5.6 DPOP とバックトラックの併用 (木の幅の制限=8) 105