修 士 論 文

先行制約付きタスクの通信遅延を考慮したマルチプロセッサスケジューリングに関する研究

指導教官　金子峰雄 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

桂 元保

2000 年 2 月 15 日

目 次

第 1章

序論

先行制約付きタスクは，各タスクをマルチプロセッサのプロセッシングエレメント (PE) へ割り当て，また同時に各タスクをコントロールステップへ割り当てることにより，マルチプロセッサ上に実装される．従って，タスク集合を高速に実行するためには，全実行時間を最小化するようなマルチプロセッサスケジューリングを求めることが重要である．ここで取り扱うマルチプロセッサスケジューリングにおいては，すべての PE は等しい性能を持ち一つの PE は同時に一つのタスクのみ実行可能であり，どの二つの PE 間にも接続があるものと仮定する．

従来のタスクスケジューリングに関する研究では，通信遅延はほとんど考慮されていない．従来の通信遅延を考慮しない先行制約付きタスクは，半順序集合 $(T, \prec)$ として表現できる．そのため，先行制約付きタスク $\mathcal{T}$ は，タスクの集合 T，T 上の半順序 $\prec$，及びタスクの処理時間関数 $W : T \to \{1, 2, \cdots\}$ で表される．$u \prec v$ は，タスク u の処理が終了するまでタスク v の処理が開始できないという制約を表現する．対象とする計算モデルでは，マルチプロセッサ P の PE 数を m とするとき，同一時刻に高々 m 個のタスクしか実行できない．このような視点からタスクのコントロールステップ $\tau : T \to \{0, 1, 2, \cdots\}$ は，$u \prec v$ ならば，$\tau(v) \leq \tau(u) + W(u)$ を満たし，任意の時刻 t に対し $\tau(v) \leq t < \tau(v) + W(v)$ となるタスク v が高々 m 個であるならば，$\mathcal{T}$ の P 上へのスケジューリングという．スケジューリングの全実行時間は，$\max_{v \in T}[\tau(v) + W(v)] - \min_{v \in T} \tau(v)$ で表される．与えられた整数 k に対して，全実行時間が k 以下になるスケジューリングが存在するか否かを判定する問題がスケジューリング問題である．この問題は $\mathcal{NP}$ 完全であることが示されている [6]．さらに，この問題はいくつかの制約条件を付加しても $\mathcal{NP}$ 完全であることが示されている [1, 8, 10]．例えば，すべてのタスクの処理時間 W を単位時間に制限してもこの問題は $\mathcal{NP}$ 完全であり，PE 数 m を 2 として，各タスクの処理時間 W を 1 または 2 に

限ってもこの問題は$\mathcal{NP}$完全のままであることが知られている[1]．一方では，ある制約条件かではこの問題は多項式時間で解けることが知られている．$m = 2$とし，各タスクの処理時間を単位時間に限ればこの問題は多項式時間で解くことが出来る[4, 5]．同様に，$\prec$が根付き木の構造をしており，各タスクの処理時間が単位時間であるならば，多項式時間で解け[1]，$\prec, W, m$に制約がある場合の発見的手法なども開発されている[7, 9]．

一方，近年におけるＶＬＳＩ技術の目覚しい発展と共に，計算機の処理能力が飛躍的に向上した．そのため，従来のスケジューリング問題では無視されていた通信遅延が全実行時間に占める割合が大きくなってきている．通信遅延を考慮したスケジューリング問題に関しては，先行制約付きタスクは閉路を含まない有向グラフ(DAG)Dで表され，二つのタスクを結ぶ有向辺(u, v)はタスクuの出力データがタスクvの入力として必要であることを表す．この問題では各タスクvの処理時間，及び各データaの通信時間はそれぞれ$W_V(v)$，$W_A(a)$で表される．通信遅延を考慮したスケジューリング問題に関しても先行制約付きタスクのグラフ構造が根付き木に制限され，$W_V(v)$と$W_A(a)$が単位時間に制限されている問題に関しての無遅延化手法とクリティカルパススケジューリング手法を用いた近似差$m - 2$の近似アルゴリズムが提案されている[2]．

本研究では先行制約関係が根付き木及びDAGで，$W_V(v)$と$W_A(a)$が単位時間に制限されている通信遅延を考慮したスケジューリング問題に関する発見的手法を提案した．

第 2章

スケジューリング問題

前章で述べたように，先行制約付きタスクを並列計算機上に高速に実装できる割り当てを求める問題は，グラフのスケジューリング問題に置き換えて考えることができる．

図 2.1 は，スケジューリング問題の簡単な例である．先行制約付きタスクは，図 2.1(a) で表すように各タスクを頂点とし，タスク間の先行制約関係を有向辺によって表すことによって，有向グラフとして表現できる．

有向辺は，始点となるタスクの処理が完了してから終点となるタスクの処理を開始することを表す．例として，図 2.1(a) に，タスク数 5 の先行制約付きタスクを示す．タスクの計算時間として，v_2,v_3 は 2，v_1,v_4,v_5 は 1 を持っている．また，有向辺 (v_1, v_2) は，タスク v_1 の処理が完了してからタスク v_2 の処理が開始できることを表す．

並列計算機は，図 2.1(b) のように各プロセッシングエレメント (PE) を頂点で表し，各 PE 間が相互接続していてデータ通信することを PE 間を結ぶことで表す．図 2.1(b) の例は，二つの PE P_1,P_2 が相互接続している．

スケジュール問題は，先行制約付きタスクを並列計算機上に実装したとき，全実行時間が最小になるようにタスクを実行する PE と実行を開始する時刻を求める，すなわち，タスクを割り当てるコントロールステップを求める問題である．

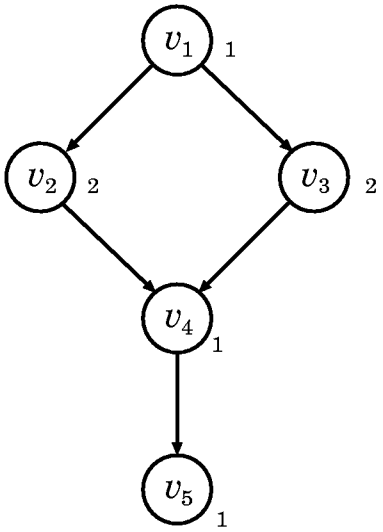
2.1　通信遅延を考慮したスケジューリング

先行制約関係にある二つのタスクを同じ PE 上に割り当てるとき，通信遅延は生じないが，これら二つのタスクを結線された異なる PE に割り当てるときには，二つのタスクを実行する間に通信遅延が生じる．例えば，図 2.1 において，先行制約関係にあるタスク v_1,v_2 が共に PE P_1 に割り当てられるならば，図 2.1(c) に示すように，v_1 はコントロール

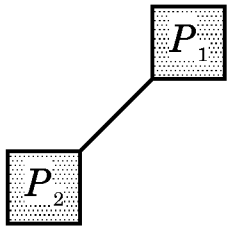
ステップ 1 で開始し，また v_2 はコントロールステップ 2 で開始し，3 で終了ことになる．しかし，タスク v_1,v_3 において，タスク v_1 が PE P_1 へ，v_3 が PE P_2 へ割り当てられるならば，v_1 の完了から v_3 の開始に至るには P_1，P_2 間のデータ通信が生じる．スケジュール問題を考えるとき，このデータ通信を考慮しなければ，図 2.1(c) に示すように v_1 の完了後，従属関係にあるタスク v_2,v_3 は同時刻に開始することができる．しかし，現実には，PE 間にはデータ通信が存在するために，通信遅延を考慮しなければならない．図 2.1(d) は，PE 間にコントロールステップと同じ単位時間の通信遅延を考慮した場合のスケジューリング結果で，PE P_2 に割り当てられている v_3 は，v_1 と同じ PE に割り当てられている v_2 より単位時間遅れて開始している．

2.2 通信遅延を考慮したスケジューリングの制約

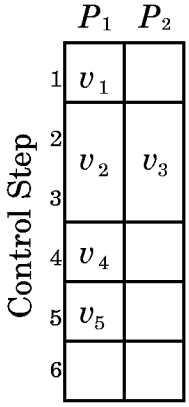
ここで，通信遅延を考慮する場合のスケジューリング問題を考えるときは次のように制約する．図 2.2 に示すように，あるタスクと先行制約関係にあるタスクが複数存在する時は，同一なコントロールステップで同一プロセッシングエレメント上では 1 つのタスクしか実行できないことを考慮しなくてはならない．それは，先行制約関係にあるタスクの実行がプロセッシングエレメント間を跨いだときには，通信遅延が発生するからである．本論文では，この通信遅延を考慮した上でスケジューリングを行い，先行制約タスクを並列計算機上へ実装した時の全実行時間を最小化するような発見的な手法を求めることを目的としている．



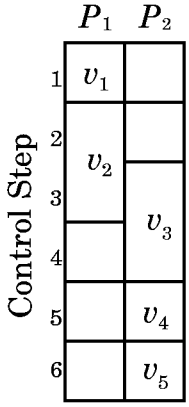
(a) タスクグラフ,



(b) プロセッシングエレメント.

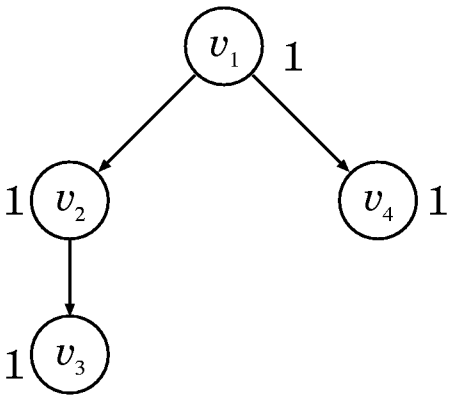


(c) 遅延を考慮しない場合の, 割り当て

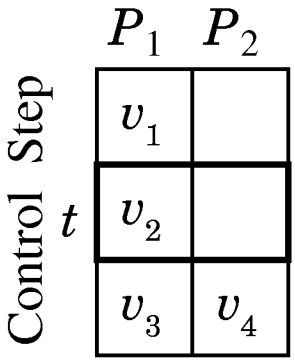


(d) 遅延を考慮した場合の. 割り当て

図 2.1: スケジューリングモデル.



(a) 先行制約付きタスク.



(b) v_1 の子供のうち，同一 CS の同一 PE 上へはタスクは一つだけ割り当てできる.

図 2.2: 通信遅延を考慮したスケジューリングの制約.

第 3章

モデルの定義

本章では，本研究で用いる並列アルゴリズムのモデルについて定義する．本研究で用いるスケジューリングモデルは従来研究で用いられているものに従う．

3.1 先行制約付きタスク

前章でも述べたように，先行制約付きタスクはn個のタスク$v_1, \ldots, v_n$を持ち，タスク間にデータの通信がある時，タスク間に有向辺を持つ有向グラフとして表現できる．この時，グラフには有向閉路，自己閉路は含まない．すべてのタスクは単位計算時間であるとする．各タスクの先行制約関係は，有向辺(v_i, v_j)で表す．(v_i, v_j)は，タスクv_iの処理が終了するまでタスクv_jの処理は開始できないという制約を表す．

3.2 並列計算機

並列計算機は，複数のプロセッシングエレメント(PE)とすべてのPE間の結線から構成される．本研究で用いるモデルでは，各PEの性能は等しく，すべての結線の通信性能も等しいと仮定する．PE数をmとする時，各々のPEを$P_1, P_2 \ldots, P_m$で表す．ここでは簡単のため，先行制約関係にある二つのタスクが異なるPEで処理されるときのPE間の通信にかかる時間もタスクを各PEで処理するためにかかる時間も単位時間であると仮定する．

3.3 スケジューリング

先行制約付きタスクのスケジューリングは，各タスク v_i を PE へ割り当てるアロケーション関数 ϕ とコントロールステップ (正の整数) へ割り当てるコントロールステップ関数 τ の対 $\langle \phi, \tau \rangle$ で表される[1]．この時，異なる二つのタスクが同時刻に同一の PE 上で処理されない．各タスク v_i の割り当てられるコントロールステップを $\tau(v_i)$ とする時，このスケジューリングモデルでは次の条件を満たす．図 3.1 に示すように (v_i, v_j) という制約がある時，v_i と v_j が同一の PE に割り当てられるならば，$\tau(v_i) + 1 \leq \tau(v_j)$ が成り立ち，v_i と v_i が異なる PE に割り当てられるならば，$\tau(v_i) + 2 \leq \tau(v_j)$ が成り立つ．

G を先行制約付きタスクとする．G のスケジューリング $\langle \phi, \tau \rangle$ が与えられた時，スケジューリングにより変換されるグラフを $G^{\langle \phi, \tau \rangle}$ で表す．また，グラフ G のコントロールステップ数を G の全実行時間 (Make Span) と呼び，MS(G) で表す．特に，$\langle \phi, \tau \rangle$ を強調するときは $\mathrm{MS}_{\langle \phi, \tau \rangle}(G)$ で表す．

3.4 先行制約タスクの制約条件

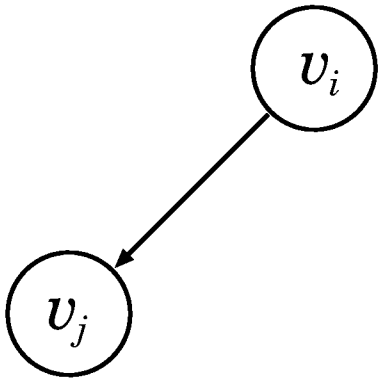
本論文では，先行制約付きタスクの制約条件として，有向閉路を含まない一般的なグラフ (directed acyclic graph, 以下 DAG3.2(a)) 及び根付き木 (3.2(b)) を取り扱う．

先行制約タスクの制約条件を DAG とした時，G で表す．DAG G に有向辺 (v_i, v_j) が存在するならば，v_i を v_j の親 (parent) と呼び，v_j を v_i の子 (child) と呼ぶ．また，各々の集合を $\mathrm{Par}(v_j)$，$\mathrm{Ch}(v_i)$ で表す．G 上に v_i から v_j への有向パスが存在するならば，v_i は v_j の祖先 (predecessor) と呼び，v_j を v_i の子孫 (successor) と呼ぶ．また，各々の集合を $\mathrm{Pred}(v_j)$，$\mathrm{Succ}(v_i)$ で表す．G のタスクの集合を $V(G)$ で表し，有向辺の集合を $A(G)$ で表す．v_i に対し，$\#\{v_j | (v_i, v_j) \in A(G)\}$ を v_i の出次数 (out-degree) と呼び，$\#\{v_j | (v_j, v_i) \in A(G)\}$ を v_i の入次数 (in-degree) と呼ぶ．

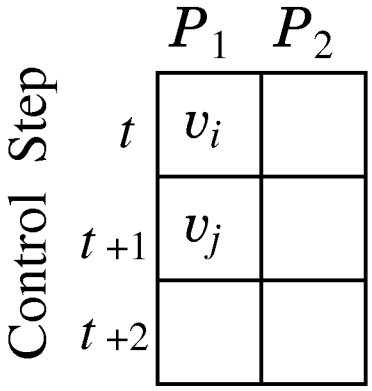
先行制約タスクの制約条件が根付き木である時，T で表す．DAG と同様に，根付き木 T に有向辺 (v_i, v_j) が存在するならば，v_i を v_j の親 (parent) と呼び，v_j を v_i の子 (child) と呼ぶ．また，子の集合を，$\mathrm{Ch}(v_i)$ で表す．T 上に v_i から v_j への有向パスが存在するならば，v_i は v_j の祖先 (predecessor) と呼び，v_j を v_i の子孫 (successor) と呼ぶ．T において，木の入次数 0 の点を根 (root) と呼び，出次数 0 の点を葉 (leaf) と呼ぶ．T のある点を v_i とした時，v_i の子孫すべてから誘導される部分木を T_i で表す．T の根から葉までのパスの長さの最大値を T の高さと呼び，$h(T)$ で表す．

---

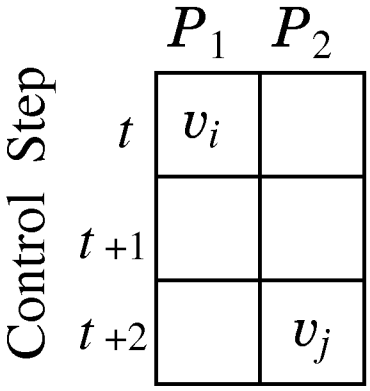
[1]‘通信遅延を考慮しないスケジューリング’は，コントロールステップ関数と半順序関係のみで表される．



(a) v_i,v_j の先行制約関係.

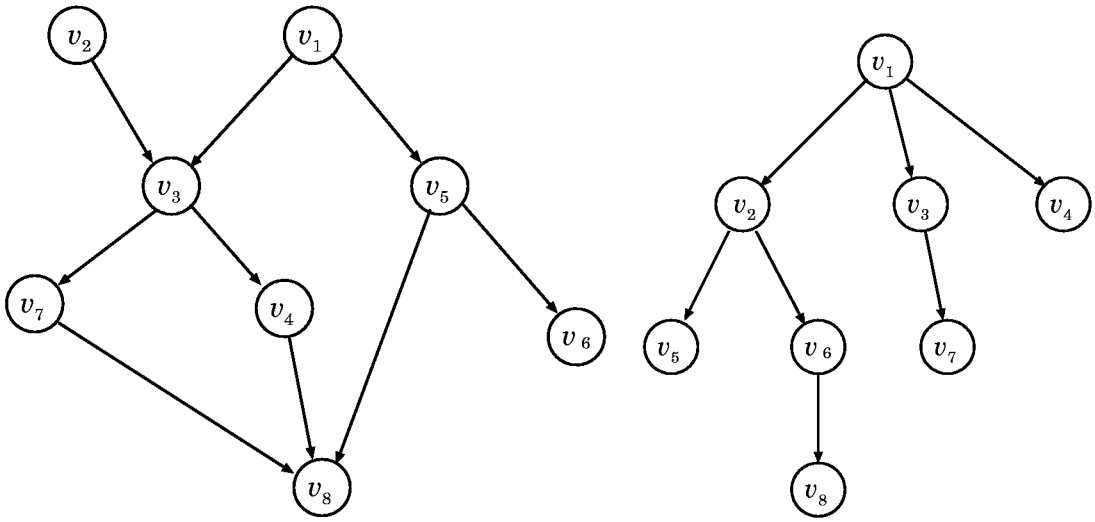


(b) v_i,v_j が同じ PE に割り当てられた場合, $t(v_i)+1 \leq t(v_j)$



(b) v_i,v_j が異なる PE に割り当てられた場合, $t(v_i)+2 \leq t(v_j)$

図 3.1: 先行制約関係にあるタスクの割り当てる PE に対するコントロールステップ.



(a) dag. (b) 根付き木.

図 3.2: 先行制約付きタスクの制約条件.

第 4章

根付き木のスケジューリング

本章では，タスクの先行制約が根付き木の構造をしている時のスケジューリング手法について考察する．[2] では，遅延を考慮したスケジューリングの解を求めるために遅延を伴う先行制約関係を表す有向グラフ T を無遅延化グラフ T^{DFT}(delay free graph 以下 DFG と呼ぶ) に変換し，DFG T^{DFT} による遅延を考慮しないスケジューリングを求めることで，元のグラフ T のスケジューリングを求める手法を提案している (図 4.1) 以下に [2] の手法を説明し，問題点を考察する．

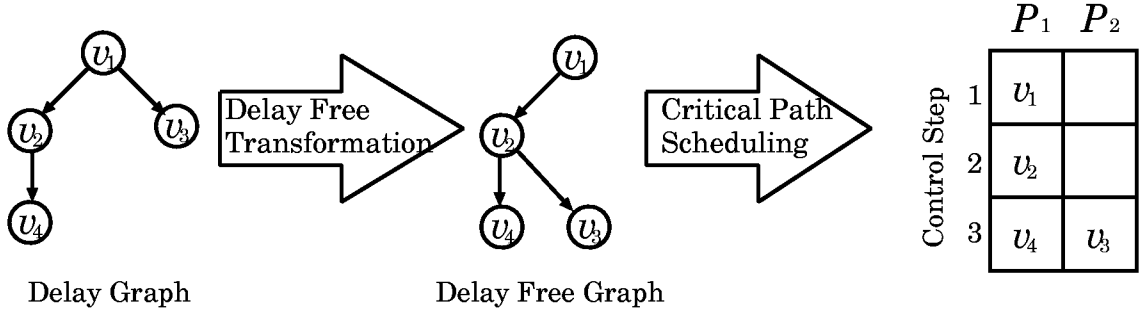
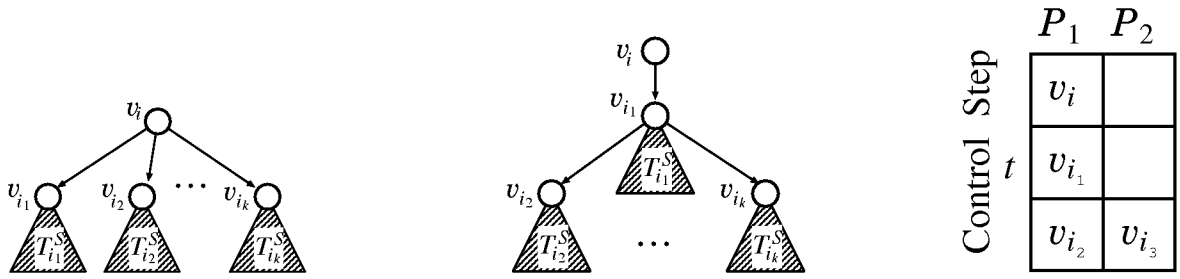


図 4.1: [2] で提案された根付き木のスケジューリングの流れ

4.1 無遅延化グラフへの変換

本節では，Varvarigou らにより提案された無遅延化グラフへの変換 (dalay free transformation 以下 DFT と呼ぶ) の手法を述べる．この手法は，部分木の高さに関して再帰的に DFG を構成する．根付き木 T の部分木のうち，v_i の子孫すべてから誘導される部分木を



(a) 遅延グラフ (delay graph). (b) 無遅延化グラフ (delay free graph). (c) 割り当て

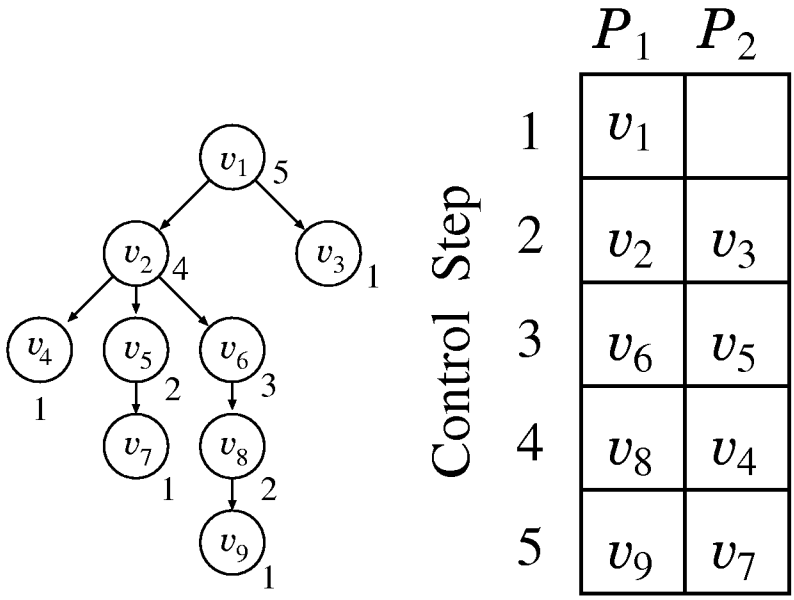
図 4.2: 無遅延化グラフへの変換 (delay free transformation).

T_i で表す．T_i が高さ 0 の部分木 (v_i1 点から成る木) ならば，それ自身を T_i^{DFT} とする．v_i のすべての子 v_j に関して T_j^{DFT} が与えられた時，T_i の DFG T_i^{DFT} は以下のように構成される．v_i の子供 $\mathrm{Ch}(v_i)$ から一つを v_i の長子 (Favored Childfc(v_i)) として選ぶ．v_i と長子 $\mathrm{fc}(v_i)$ 以外の子供 $\mathrm{Ch}(v_i)\backslash\mathrm{fc}(v_i)$ とを結ぶ有向辺を取り除き，長子から他の子へ有向辺を結ぶ．v_i と長子以外の子供の間の有向辺を長子と長子以外の子供との間の有向辺として付け変える操作は，2.2 で述べたあるタスクと先行制約関係にあるタスクが複数存在する時は同一なコントロールステップで同一プロセッシングエレメント上では 1 つのタスクしか実行できないことを有向グラフで表現するための操作であり，有向辺を付け替えて生成された無遅延化グラフでは付け変える前の遅延グラフでの各タスク間の先行制約関係は保持している．図 4.2 では，v_{i_1} を長子に選んだ場合の T_i^{DFT} の例を示している．この操作を再帰的に繰り返すことにより，T の無遅延化グラフ T^{DFT} が構成される．

4.2 クリティカルパススケジューリング

根付き木の遅延を考慮しないスケジューリングに関しては，[1] ではクリティカルパス法則を用いて最適解を求めるアルゴリズムを構成した．そのアルゴリズムは以下の通りである．各タスク v_i に，v_i から葉までの最長有向パスの長さを v_i の重みとしてつける．コントロールステップの 1 ステップに根を割り当てる．t までにすべての祖先が割り当てられたタスクのうち，最も重みが大きいタスクから順に割り当てる．図 4.3 に例を示す．図 4.3(a) の各タスクに付されている数値は，各タスクから葉までの最長有向パスの長さであり，このアルゴリズムで与えられる重みである．まず，グラフの根である v_1 がコントロールステップ 1 に割り当てられる．次に，祖先が割り当てられたタスク v_2,v_3 がコントロールステップ 2 に割り当てられる．さらに，祖先が割り当てられたタスクの中から重みが大

きいタスク v_5 と v_6 がコントロールステップ 3 に割り当てられる．以降，すべてのタスクが割り当てられるまでこの操作を行う．



(a) 先行制約付きタスク.　　(b) スケジューリング.

図 4.3: クリティカルパス則によるスケジューリング.

4.3 高さ優位の DFT

T^{DFT} に関して，はじめに述べた遅延を考慮しないスケジューリング (時刻関数)τ が与えられた時，T のスケジューリングは次のように得られる．$\tau(v_i) = 1$ であるタスクに関しては，スケジューリングのアロケーション関数 ϕ を各タスクが異なる PE に割り当てられるように任意に決める．コントロールステップ t までに割り当てられたタスクを処理する PE が与えられた時，$\tau(v_i) = t + 1$ のタスクのうち，その親がコントロールステップ t に割り当てられてるならば，v_i はその親と同じ PE に割り当てる．他のタスクは，そのコントロールステップに於いてまだタスクが割り当てられていない PE に割り当てる．[2] では，まず，v_i の長子 $\mathrm{fc}(v_i)$ を v_j として，T_j^{DFT} の高さ $h(T_j^{\mathrm{DFT}})$ が最大である子の一つを任意に選び無遅延化グラフへの変換を行い T_i^{DFT} を構成する．この操作を再帰的に行い元の遅延グラフ T の無遅延化グラフ T^{DFT} を構成する．さらに，構成された T^{DFT} に対し，[1] のクリティカルパス手法を適用することにより，T^{DFT} のスケジューリングを行っている．

具体例を図4.4に示す．図4.4(a)に示すような根付き木構造の先行制約付きタスクTが与えられたとする．まず，v_2を根とする部分木T_{v_2}に対して無遅延化変換を行う．この例では，v_2の子供を根とする各部分木の高さが最も大きいv_6がv_2の長子として選ばれ，有向辺を付け替える(図4.4(b))．さらにv_1を根とする部分木T_{v_1}に対して無遅延化変換を行う．v_1の子を根とする部分木の高さが最も大きいv_2がv_1の長子として選ばれ，有向辺を付け替える．T_{v_1}=Tなので，Tの無遅延化変換が完了し，Tの無遅延化グラフT^{DFT}が生成される(図4.4(c))．次に，生成された無遅延化グラフT^{DFT}に対してクリティカルパス手法を適用する．図4.4(c)の各タスクには，各タスクから葉までの最長有向パスの長さをタスクの重みとして付してある．まず，コントロールステップ1でグラフの根であるタスクv_1がPE P_1に割り当てられる．次に，コントロールステップ2で祖先がすでに割り当てられたタスクv_2がv_2と同じPE P_1に割り当てられる．v_2はv_1の長子でありv_1と同じPEに割り当てられるので，通信遅延は発生しない．さらに，コントロールステップ3では，祖先がすでに割り当てられたタスクがPEの数以上あるので，その中で重みが大きい方からm個選ばれる．すなわち，重み1が付されているタスクv_6とv_3が選ばれる．割り当ては，まず前のコントロールステップに親が割り当てられているv_6から行い，PE P_1に割り当てられる．そして，残ったPEにv_3が割り当てられる．v_3は元のグラフTではv_1の子供であり，同じ子供であるJ_2よりも1ステップ遅れてP_2に割り当てられることから，通信遅延が考慮されていることになる．

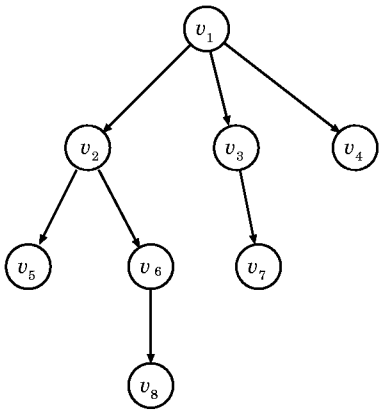
アルゴリズムを図4.5に示す．

このようにして得られたスケジューリングはTの遅延を考慮したスケジューリングであることは容易にわかる．また，MS(T)の最小値を与えるようなTのDFT T^Sが存在することが知られている[2]．しかし，そのようなT^Sを求める手法は見つかっていない．[2]では，v_iの長子v_jとして，T^S_jの高さが最大である子の一つを任意に選び，T^Sを構成し，Hu [1]により提案されたクリティカルパス手法をT^Sに適用することにより，T^Sのスケジューリングを構成している．先にも述べたが，Huによる手法はT^Sに関する‘通信遅延を考慮しないスケジューリング’の最適解を与える．

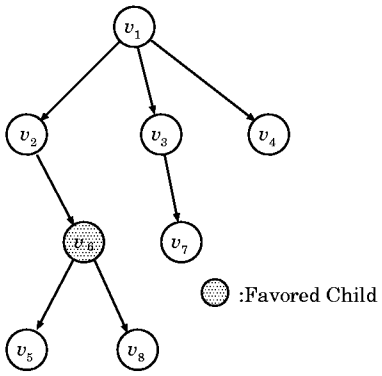
4.4 問題点

この節では，Varvarigouらの手法で求めたT^Sから得られる解と最適解との差が任意の整数となるような根付き木Tの例を挙げる．Varvarigouらは彼らの求めたスケジューリングと最適解との差が高々$m-2$であることを主張している．任意のkに対し，図4.6(a)により与えられる木は適当なmとxを与えることにより，Varvarigouらのアルゴリズムを適

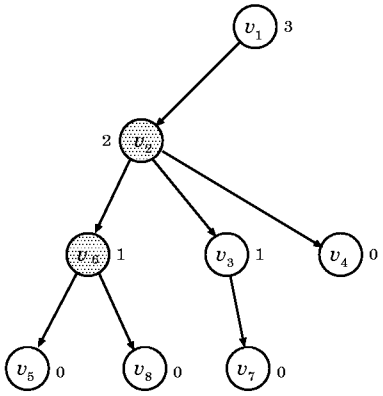
用して得られる解と最適解との全実行時間の差が k となる例である．図 4.6(b) は $k = 2$ の例である．このとき，$x = 23$，$m = 6$ とする．図 4.6(b) の最適解は全実行時間が図 4.6(a) に示すように 9 であるが，[12] の手法を適応すると図 4.6(b) に示すように 11 となる．(図 4.7 において，太枠で囲まれたタスクは同一 PE 上で処理されなければならない．)



(a) 遅延グラフ T.



(b) 無遅延化変換の途中.



(c) 無遅延化グラフ T^{DFT}.

Control Step	P_1	P_2
1	v_1	
2	v_2	
3	v_6	v_3
4	v_8	v_4
5	v_7	v_5

(d) クリティカルパススケジューリング結果.

図 4.4: [2] の手法の実行例.

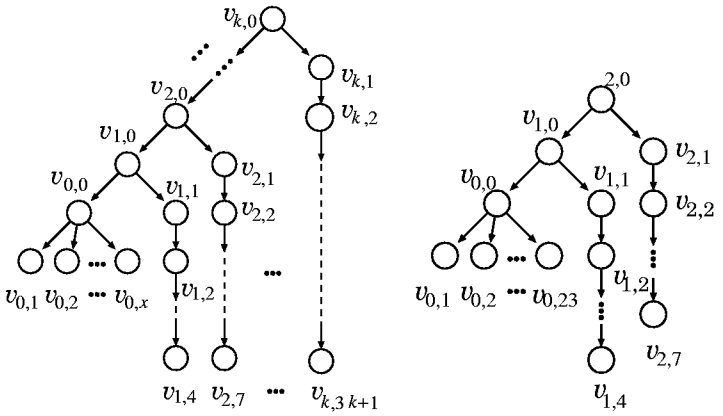
program algo T;
begin
{
　$T^{\mathrm{DFT}} = \mathrm{dft}(T)$; /* 無遅延化変換 */
　$\mathrm{cp}(T^{\mathrm{DFT}})$;
}
end.

subroutine $\mathrm{dft}(T_i)$; /* 無遅延化変換 */
{
　if(root(T_i) の子供 v_j を根とする部分木 T_j
　　の DFT が完了していない)
　{
　　$T_j^{\mathrm{DFT}} = \mathrm{dft}(T_j)$; /* T_j について無遅延化変換を行う */
　}
　/* 無遅延化グラフの根 v_j で最も高さが高いものを T の根 v の長子とし $\mathrm{fc}(v)$ とする． */
　$\mathrm{fc}(v) = \{v_j | \max_{j=1,\cdots,k} h(T_j^{\mathrm{DFT}})\}$;
　$\mathrm{Ch}(v) \backslash \{\mathrm{fc}(v)\}$ を $\mathrm{fc}(v)$ の子供とし，有向辺をつけ替える．
}

subroutine $\mathrm{cp}(T)$; /* :クリティカルパススケジューリング */
{
　v_i から葉までの最長有向パスの長さを w_{v_i} とする．
　T の根からコントロールステップへ割り当てる．
}

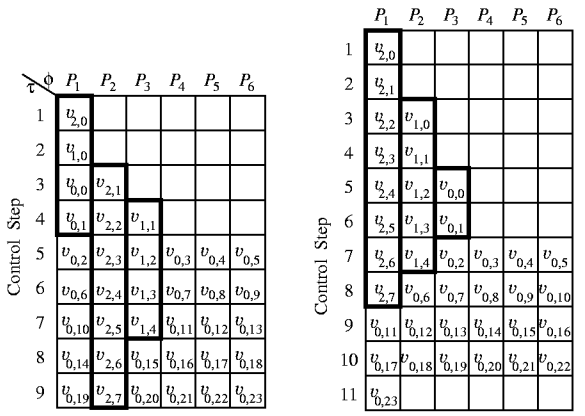
T:遅延を考慮する根付き木
T^{DFT}:無遅延化グラフ (delay free graph)
$\mathrm{cp}(T)$:クリティカルパススケジューリング
$\mathrm{dft}(T)$:無遅延化変換 (delay free transformation)
$\mathrm{root}(T_i)$:T_i の根
$\mathrm{fc}(v)$:v の長子 (favored child)
$\mathrm{Ch}(v)$:v の子供 (children) の集合

図 4.5: 高さ優位の DFT アルゴリズム



(a) T'_k. (b) T'_2.

図 4.6: 最適解との差が整数 k になる入力 T'_k.



(a) T'_2 の最適解. (b) [2] を適用した解.

図 4.7: T'_2 の最適解と [2] による解の比較.

第 5章

重み付けによる優先順位割当

この章では，単位通信遅延を持つ根付き木のスケジューリングについて，各タスクに優先順位 (重み) を付け，重みの大きいタスクから順に割り当てを決め，それと同時に重みを付け替える動的な重み付け手法を用いて，DFT 手法よりも最適解に近似的な解をもたらす手法を提案する．

5.1 最終コントロールステップからの割当

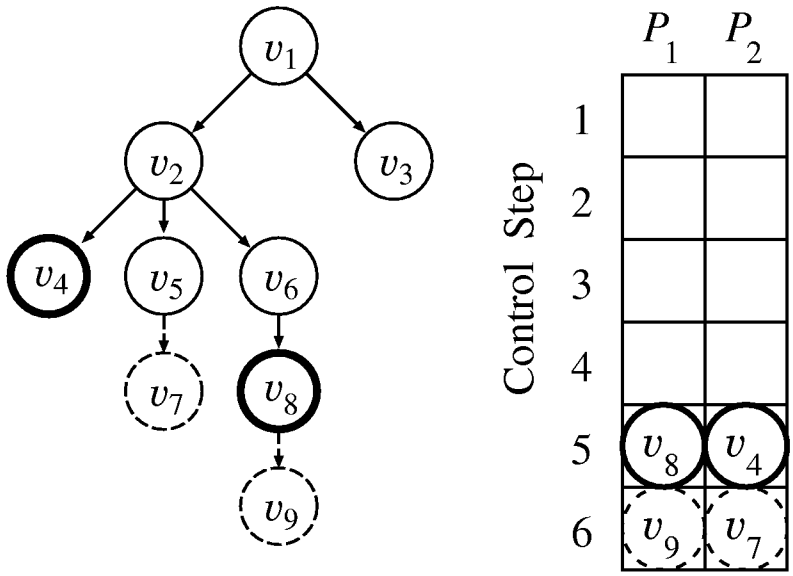
我々の用いる手法は，子がすべて割り当てられたタスクを最終コントロールステップ t_{last} 側から順に割り当てる．

コントロールステップ $t+1$ から t_{last} の間に割り当てられたタスクを元の入力 T から除いて得られる木を $S_t(T)$ とする．コントロールステップ t においてタスク v_i は，コントロールステップ $t+1$ に高々一つの子 $\mathrm{fc}(v_i)$ が割り当てられていて他の子はコントロールステップ $t' \geq t+2$ に割り当てられているならば，割り当て可能なタスクと呼ぶ．ただし，ある一つのタスク $v_{i'}$ が複数の子を持ち，そのすべてが葉である時はそのうちの一つは $v_{i'}$ の長子 $\mathrm{fc}(v_i)$ とし，他の子供 $\mathrm{Ch}(v_i)\backslash\mathrm{fc}(v_i)$ を割り当て可能とする[1] ．タスク v_i がコントロールステップ t に割り当てられていて，v_i の子のうちの一つ v_j がコントロールステップ $t+1$ に割り当てられているならば，v_i と v_j は同一の PE に割り当てられなければならない．コントロールステップ t に割り当てるタスクは以下のように決める．まず，$S_t(T)$ の各タスクに次節で述べる重み付けアルゴリズムに従い優先順位を付し，割り当て可能なタスクのうち優先順位の高いものから順に m 個のタスクをコントロールステップ t に割り当てる．

[1] コントロールステップ t で $v_{i'}$ の複数の子が同時に割り当てられたとしても，$v_{i'}$ は $t-1$ に割り当てることが出来ないので，他の葉を優先して割り当てる．

このコントロールステップで割り当てられたタスクを $S_t(T)$ から除いたものが $S_{t-1}(T)$ である．この操作を $S_t(T)$ が空になるまで繰り返す．重み付けを省略した例を図 5.1 に示す．最終コントロールステップ $t_{\rm last}$ で割り当て可能なタスクは v_3, v_4, v_7, v_9 である．そのうち v_7 と v_9 を割り当て，T から v_7 と v_9 を除去して得られる木を $S_{t_{\rm last}-1}(T)$ とする．$t_{\rm last}$ で割り当て可能なタスクは v_3, v_4, v_5, v_8 である．このうち v_4 と v_8 を割り当てる．v_8 はその長子 v_9 と同一の PEP_2 に割り当てられる．以下同様にして，図 6(b) のスケジューリングが得られる．結果として得られるスケジューリングの全実行時間 MS(T) は 6 なので，$t_{\rm last} = 6$ とする．

アルゴリズムを図 5.5 に示す．



(a) 入力グラフ T　　(b) スケジューリング

図 5.1: スケジューリング例．

5.2　タスクへの重み付け

5.1 において，$S_t(T)$ の葉からどの葉を m 個選択するかを決めるための優先順位を重みとして定める．その重み付けの手続きを述べる．

まず，各タスクから葉までの最大パスの長さを把握するために，以下のようにタスクに仮の重みを付ける．スケジューリングのステップ t におけるグラフ $S_t(T)$ の葉には，仮の重みとして 1 を付ける．もし，タスク v の子が一つ $|{\rm Ch}(v)| = 1$ ならば，v の仮の重みと

してvの子の仮の重みに1加えた値を付け，タスクvの子が二つ以上$|\mathrm{Ch}(v)| > 1$ならば，子の中で仮の重みが最大である値に1を足した値をvの仮の重みとして付ける．さらに，仮の重みが最大であるvの子供が複数存在すれば，その最大値に2を足した値をvの仮の重みとする．

次に，有向辺に重みをつける．有向辺の重みは，有向辺の終点となるタスクに付されている仮の重みによって付けられる．まず，vに対して，vの子が一つ$|\mathrm{Ch}(v)| = 1$ならば有向辺の重みとして1をつける．vの子が二つ以上$|\mathrm{Ch}(v)| > 1$ならば，$\mathrm{Ch}(v)$の中からvの長子$\mathrm{fc}(v)$を次のように決める．もし，$\mathrm{Ch}(v)$に仮の重みが最大であるタスクが1つあれば，そのタスクを長子$\mathrm{fc}(v)$とする．もし，仮の重みが最大であるタスクが複数存在すれば，その中で子孫数が最大であるタスクを長子$\mathrm{fc}(v)$とする．仮の重みが最大で子孫数も最大である子が複数存在すれば，長子$\mathrm{fc}(v)$はその中で任意に決定する．そして，始点がvで終点が長子$\mathrm{fc}(v)$である有向辺$(v, \mathrm{fc}(v))$に重み$1 - \varepsilon$を付け，始点がvで終点が長子以外の子$\mathrm{Ch}(v)\backslash\mathrm{fc}(v)$である有向辺$(v, \mathrm{Ch}(v)\backslash\mathrm{fc}(v))$に重み$2 - \varepsilon$を付ける．ただし，$\varepsilon$は1より十分小さい正の実数とする．有向辺の重みに$\varepsilon$を適用するのは，以下の理由からである．有向辺の重みで特に整数部の値は，有向辺の始点と終点に対応するタスクの最短のコントロールステップの差を表している．あるタスクv_iが子供を1つだけ持つ場合，最短のコントロールステップの差は，同じPEに割り当てられた時であり1コントロールステップなので，有向辺の重みの整数部は1となる．また，v_iが子供を複数持つ場合，v_iの長子との最短のコントロールステップの差は同じPEに割り当てられた時なので，対応する有向辺の重みは1，長子以外の子供は長子よりも前に割り当てられないのでコントロールステップの差は最短でも2になり，対応する有向辺の重みは2を付ける．この重み付けは，2.2の通信遅延を考慮したスケジューリングの制約条件に適応するための操作である．さらに，各タスクの重みは，グラフの根の重みを0として根とタスクv_iを結ぶ有向パスに含まれる有向辺の重みを加算した値を付ける．すなわち，タスクの重みは各タスクと根の最短のコントロールステップの差を表す．また，タスクの重みの整数部分は，各タスクと根の有向パスの長さも表している．タスクの重みの整数部分は，従来手法の無遅延化変換によって生成された無遅延化グラフの各タスクと根の有向パスの長さと対応する．5.1で述べたように，我々の手法は子がすべて割り当てられたタスクを最終コントロールステップから割り当てを行う．タスクの重みが大きいタスクは，タスクと根のコントロールステップの差が大きいことなので，最終コントロールステップからの割り当てでは最初に割り当てが行われることになる．

しかし，重みが整数部分だけだと都合の悪い場合が存在する．図5.2は，重みの整数部分が等しいタスクにおいて，割り当てるタスクの選び方によって全実行時間に差が生じる

例である．図5.2に示すグラフの特徴は，v_1の子供を根とする部分木の構造として，部分木の高さは小さいが部分木の枝分かれの回数が多い構造と，部分木の枝分かれの回数は少ないが高さが大きい構造をしているところであり，それぞれの葉の重みの整数部が等しい．この例では，v_2の長子以外の子供v_4，v_5，v_6，v_7，v_8，v_9と，根との有向パスの長さが最大であるタスクv_{13}の重みの整数部分の値が等しく4である．重みの整数部分だけで優先順位を決定すると，図5.2(a)と(b)の2通りの割り当て方が存在する．例では，図5.2(a)のように，高さが大きい部分木の葉を優先する割り当て方が望ましい．タスクの重みの整数部分が等しい時，部分木の枝分かれ回数で重みに差を生じさせるために，1より十分小さい正の実数εを導入し，高さが高い部分木の葉の優先順位が大きくなるようにした．

図5.3に，タスクへの重み付けの例を示す．図5.3(a)は，各タスクに括弧で付してある仮の重み及び有向辺に重みが付いた状態である．まず，v_3, v_4, v_7, v_9に仮の重み1を付ける．例えば，タスクv_8は子を1つ持つので仮の重みとして2を付ける．またv_2は子を3つ持ち仮の重みが最も大きいv_6をv_2の長子とし，v_2の仮の重みは4となる．

図5.3(b)は，(a)で付けた重みを基に，根の重みを0として各タスクへ重みを付ける．コントロールステップtへの割り当ては，$S_t(T)$の葉の重みを比較し，最大重みを持つタスクからm個選択して割り当てる．割り当てたタスクを$S_t(T)$から除き，生成された$S_{t-1}(T)$に対してアルゴリズムを実行する．

アルゴリズムを図5.4に示す．

5.3 評価

評価は，次の2つの方法で提案手法と[2]のアルゴリズムを実行して得られるスケジューリングの全実行時間を比較し，提案手法の有効性を確認した．

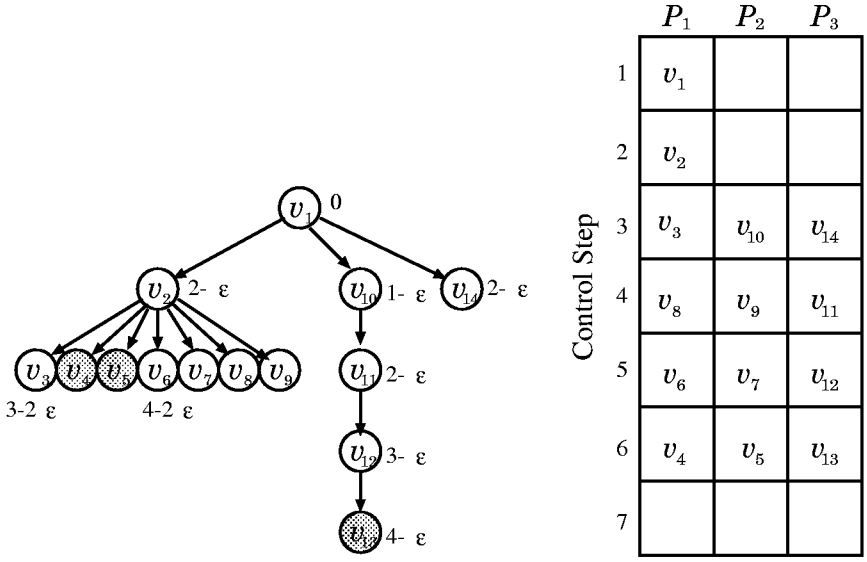
5.3.1 任意の根付き木に対する比較

まず，任意に生成した根付き木に対して，全実行時間の比較を行った．根付き木の生成は，図5.6に示すようにタスクを既成のグラフに加え，加えたタスクと先行制約関係を持たせるタスクを一様乱数により既成のグラフから決定して生成した．実験は，まずタスク数$n = 100$と$n = 500$の根付き木をランダムに100個生成する．生成された根付き木に対して，従来手法と提案手法のアルゴリズムをPE数を$m = 3$から$m = 30$まで変化させて実装し全実行時間を求め，従来手法と提案手法の全実行時間の差を求めた．表5.1は$n = 100$，表5.2は$n = 500$の実装結果を示す．各表において，各行は根付き木のサンプ

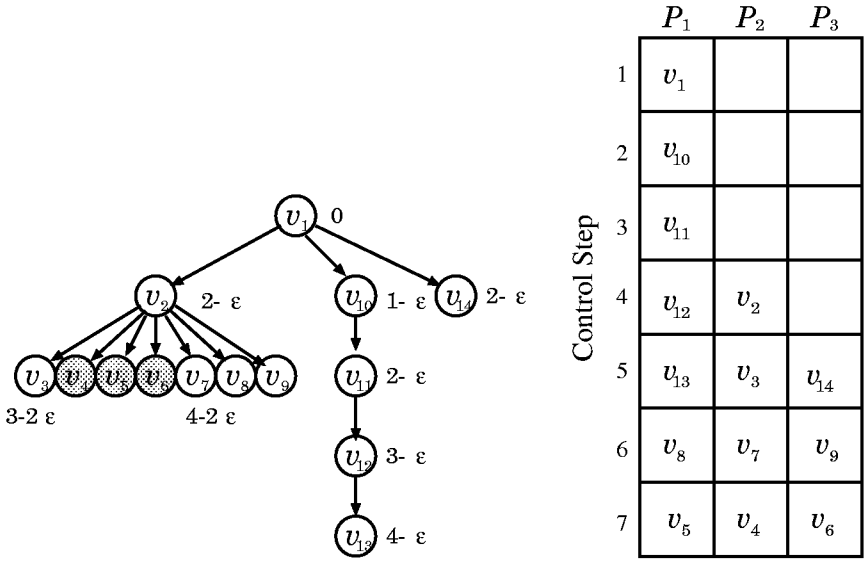
ルナンバーを表し，各列はPE数を表す．表内の数値は，従来手法による全実行時間と提案手法による全実行時間との差を表す．サンプルとPE数の全組み合わせの約12%で全実行時間に差が生じた．しかし，2以上の差は生じなかった．以上から，提案した重み付けによる手法は，全実行時間がどの入力に対しても従来手法と同じかあるいはより良い結果が得られた．

5.3.2 図4.6に対する比較

次に，図4.6に示した構造の根付き木に対する実験結果を示す．図4.6において，$n = 2, 3, 4, 5$の場合について提案手法と[2]のアルゴリズムを実行して全実行時間の比較を行った．その結果を表5.3に示す．表5.3に示すように，[2]による手法で最適解との差がnとなる入力に対しても，提案手法は最適解を与えることが確認された．数値は全実行時間を表し，$n = 2, 3, 4, 5$に対し全実行時間の差を比較している．[2]の手法は，nにより図4.6のタスク数が増大するに従って全実行時間が増大するのに対し，提案手法の全実行時間は変化しない結果が得られた．

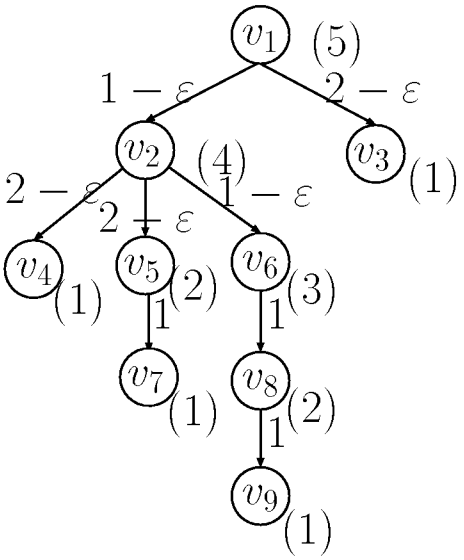


(a) v_4,v_5,v_{13} を優先して割り当て.

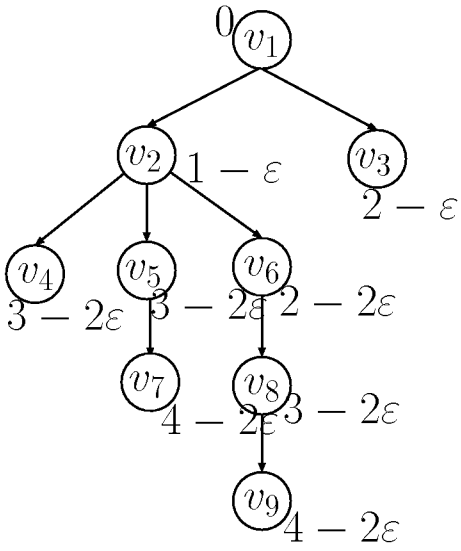


(a) v_4,v_5,v_6 を優先して割り当て.

図 5.2: 整数重みが等しい時の優先順位.



(a) タスクの仮の重み,
有向辺の重みを添付.
(カッコ内はタスクの仮の重み)



(b) タスクの重みを添付.

図 5.3: 重み付け例.

```
program add_weight_for_tree G;
integer t = 1;
begin
  while(G ≠ ∅)
  {
    tmp_w_{Leaf(G)} = 1;
    while(tmp_w_{rt(G)} が決まるまで実行)
    {
      if(|Ch(v)| = 1)
      {
        w_{(v,Ch(v))} = 1;
        tmp_w_v
        = w_{(v,Ch(v))} + tmp_w_{Ch(v)}
      }else{
        Fc(v)
        = {Ch(v)| max(tmp_w_{Ch(v)})};
        if(|Fc(v)| > 1)
        {
          fc(v)
          = {Fc(v)| max(succ_{Fc(v)})};
        }else{
          fc(v) = Fc(v)
        }
        w_{(v,fc(v))} = 1 − ε
        w_{(v,Ch(v)\{fc(v)})} = 2 − ε;
        tmp_w_v
        = max{w_{(v,Ch(v))} + tmp_w_{Ch(v)}};
      }
    }
    w_{Root(G)} = 0;
    while(∀v)
    {
      w_{Ch(v)} = w_v + w_{(v,Ch(v))};
    }
    Allocation(G, t);
    t++;
  }
end.
```

Leaf(G)：グラフ G の葉の集合
rt(G):G の根
fc(v)：v の長子
Fc(v)：v の長子の候補の集合
tmp_w$_v$：v に付けられる仮の重み
$w_{(v_1,v_2)}$：有向辺 (v_1,v_2) の重み
w_v：v の重み
succ$_v$：v の子孫数
v_{cd}^x:x 番目に重みが大きい割り当て可能なタスク

図 5.4: 重み付けアルゴリズム

```
function allocation(G, t); /* コントロールステップに割り当てる関数 */
begin
   while(∀Leaf(G))
   {
      while(x = 1; x ≤ m; x++)
      {
         v_cd^x = max^x{w_Leaf(G)};
      }
      while(x = 1; x ≤ m; x++ and x = 1; i ≤ m; i++)
         if(Alloc(P_i, t + 1) == Ch(v_cd^x) /* v_cd^x の子が前の t で割り当てられているならば*/
         {
               v_cd^x を Alloc(P_i, t) に割り当てる. ;
         }
         まだ割り当てられていない v_cd^x を他の P_i に割り当てる. ;
      }
   }
end.
```

図 5.5: 割り当てアルゴリズム

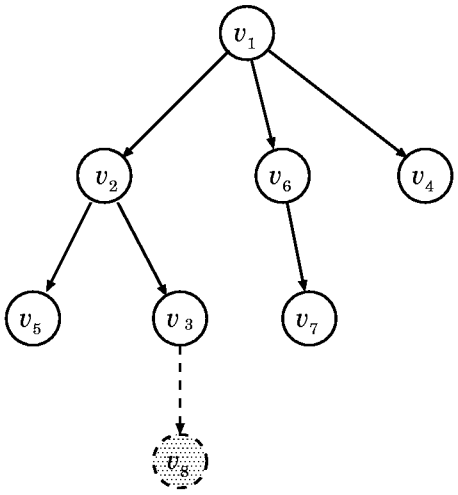


図 5.6: 根付き木の生成 (v_8 が加えられたところ)

比較　dft-weight　　N=100

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
No1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No2	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
No3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No4	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No7	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
No10	0	0	0	0	1	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
No11	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0
No12	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No14	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No15	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No16	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No17	0	1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No19	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
No20	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No22	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No24	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No29	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
No33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No36	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No37	0	0	1	0	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No38	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No41	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No42	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No43	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No44	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No47	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No49	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No51	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No56	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No57	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
No58	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No61	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No62	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
No63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No64	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No65	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No67	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No68	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No70	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No74	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No75	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No79	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
No80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No81	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No83	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No85	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No87	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
No88	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No89	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No92	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No93	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
No94	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No95	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No96	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No97	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No98	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
No100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ratio	0.12																											

表 5.1: 実装結果 $n = 100$

比較　dft-weight　　N=500

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
№1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№2	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	0	0
№3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№5	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0
№6	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
№7	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	1	1	1	0	0	1	0	1	0
№8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1	0	1	1
№10	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
№11	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
№13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
№14	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№15	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
№16	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
№17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
№18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0	0
№20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
№21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№22	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
№23	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0
№25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№26	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№27	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	0
№28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
№29	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№30	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
№33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№34	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№35	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
№36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№37	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
№38	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
№39	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0
№40	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№42	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
№43	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
№44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0
№45	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
№46	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0
№47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0
№48	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0
№49	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№51	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
№52	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
№53	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
№54	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
№55	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0
№56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
№57	0	0	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
№58	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0
№59	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
№60	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
№61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
№62	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1
№63	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
№64	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
№65	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
№66	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
№67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
№68	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0
№69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№70	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0
№71	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
№72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№74	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
№75	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
№77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
№78	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0	0
№79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
№80	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0
№81	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
№82	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№83	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
№84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
№85	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
№86	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№87	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
№88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
№89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0
№90	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	1	0
№91	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1	0	0
№93	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
№94	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
№95	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
№96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
№97	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№98	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№99	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
№100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ratio	0.12																											

表 5.2: 実装結果 $n = 500$

スケジューリング	$k=2, m=6$	$k=3, m=10$	$k=4, m=15$	$k=5, m=21$
提案手法	9	12	15	18
従来手法	11	15	19	23

表 5.3: 図 4.6 の実装結果

第 6章

一般的な先行制約付きタスクへの応用

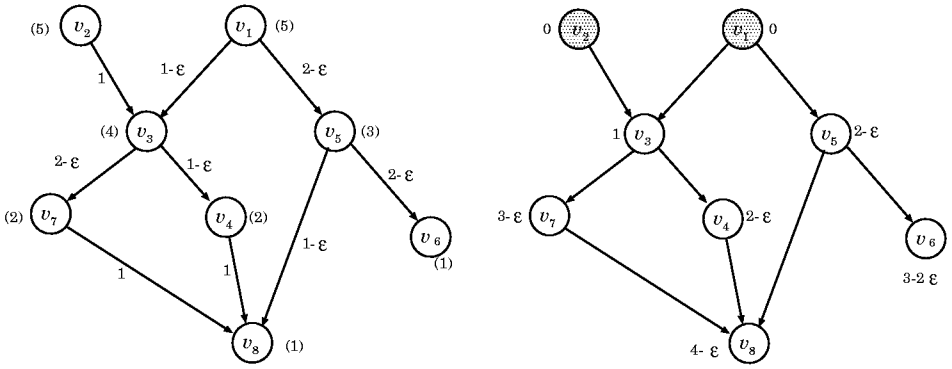
本章では，前章で提案した根付き木に対する提案手法を元に，単位通信遅延を考慮した一般的な先行制約付きタスク (DAG) に対するスケジューリングについて考察する．

6.1 アルゴリズム

提案するDAGに対するスケジューリングは，根付き木と同様に子孫を持たないタスクを最終コントロールステップ t_{last} から順に割り当てる．コントロールステップ $t+1$ から t_{last} の間に割り当てられたタスクを元の入力 G から除いて得られるDAGを $S_t(G)$ とする．割り当てるタスクの優先順位も，根付き木のスケジューリングと同様に祖先を持たないタスクからの重み付けを毎コントローステップ行い，その重みを用いて優先順位を決定する．また，タスクの重みが等しい時，優先するタスクは任意に決めず，この手法では子孫を持たないタスクからの重み付けも行い，その重みで優先順位を決定する．以下に提案手法の流れを示し，説明する．

スケジューリングの流れ

1. $S_t(G)$ の祖先を持たないタスクから重みを付ける．(6.1.1)
2. $S_t(G)$ の子孫を持たないタスクから重みを付ける．(6.1.2)
3. $S_t(G)$ の子孫を持たないタスクからコントロールステップへ割り当てる．(6.1.3)
4. $S_t(G)$ のから割り当てたタスクを除いたDAGを $S_{t-1}(G)$ として，$S_{t-1}(G)$ を step 1 から実行する



(a) タスクの仮の重み，
有向辺の重みを，
子孫を持たないタスクから添付．
(カッコ内はタスクの仮の重み)

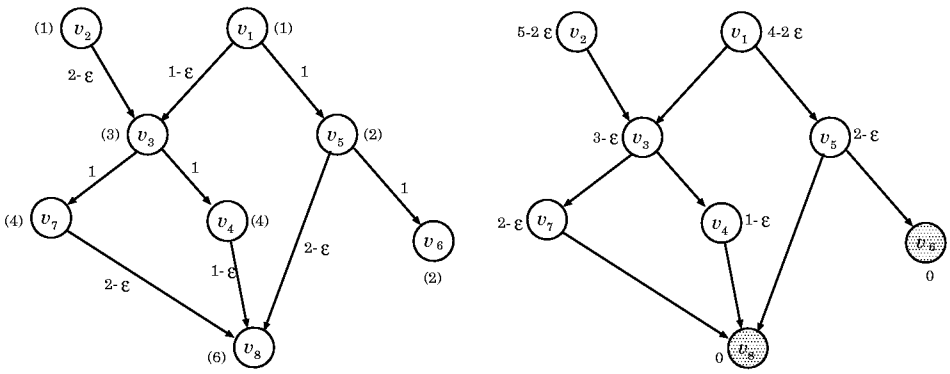
(b) タスクの重みを，
祖先を持たないタスクから添付．

図 6.1: 祖先を持たないタスクからの重み付け．

6.1.1 祖先を持たないタスクからの重み付け

$S_t(G)$ の子孫を持たないタスクから，どのタスクを m 個選択するかを決めるための優先順位を重みとして定める．まず，各タスクから子孫を持たないタスクまでの最大パスの長さを把握するために，以下のようにタスクの仮の重みを付ける．スケジューリングのステップ t における DAG$S_t(G)$ の子孫を持たないタスクには，仮の重みとして 1 を付ける．もし，タスク v の子が一つ $|\mathrm{Ch}(v)| = 1$ ならば，v の仮の重みとして v の子の仮の重みに 1 を加えた値を付け，タスク v の子が二つ以上 $|\mathrm{Ch}(v)| > 1$ ならば，子の中で仮の重みが最大である値に 1 を加えた値を v の仮の重みとして付ける．さらに，仮の重みが最大である v の子供が複数存在すれば，その最大値に 2 を加えた値を v の仮の重みとする．

次に，有向辺に重みをつける．有向辺の重みは，有向辺の終点となるタスクに付されている仮の重みによって付けられる．まず，v に対して，v の子が一つ $|\mathrm{Ch}(v)| = 1$ ならば有向辺の重みとして 1 をつける．v の子が二つ以上 $|\mathrm{Ch}(v)| > 1$ ならば，$\mathrm{Ch}(v)$ の中から v の長子 $\mathrm{fc}(v)$ を次のように決める．もし，$\mathrm{Ch}(v)$ に仮の重みが最大であるタスクが一つあれば，そのタスクを長子 $\mathrm{fc}(v)$ とする．もし，仮の重みが最大であるタスクが複数存在すれば，その中で子孫数が最大であるタスクを長子 $\mathrm{fc}(v)$ とする．仮の重みが最大で子孫数も最大である子が複数存在すれば，長子 $\mathrm{fc}(v)$ はその中で任意に決定する．そして，始点が v で終点が長子 $\mathrm{fc}(v)$ である有向辺 $(v, \mathrm{fc}(v))$ に重み $1 - \varepsilon$ を付け，始点が v で終点が長子以外の子 $\mathrm{Ch}(v)\backslash\mathrm{fc}(v)$ である有向辺 $(v, \mathrm{Ch}(v)\backslash\mathrm{fc}(v))$ に重み $2 - \varepsilon$ を付ける．ただし，ε は



(a) タスクの仮の重み,
有向辺の重みを,
祖先を持たないタスクから添付.
(カッコ内はタスクの仮の重み)

(b) タスクの重みを,
子孫を持たないタスクから添付.

図 6.2: 子孫を持たないタスクからの重み付け.

1 より十分小さい正の実数とする．図 6.1(a) は，各タスクに括弧で付してある仮の重み，及び有向辺に重みがついた状態である．まず，v_6,v_8 に仮の重み 1 を付ける．例えば，v_4，v_7 は子を一つ持つので，仮の重みとしてそれぞれ 2 を付ける．また，v_5 は子を二つ持ち，子の仮の重みは等しいので，v_5 の仮の重みは子の仮の重みに 2 を加えた 3 を付ける

さらに，$S_t(G)$ の祖先を持たないタスクから以下のように重みを付ける．まず祖先を持たないタスクの重みを 0 とする．各タスク v の重みは，v の親 $\mathrm{Par}(v_j)$ の重みと対応する有向辺 $(\mathrm{Par}(v_j), v)$ の重みの和を付ける．もし，図 6.4 のように，v の親 $\mathrm{Par}(v_j)$ が複数存在する ($|\mathrm{Par}(v_j)| > 1$) ならば，各親の重みと対応する有向辺の重みの和の最大値を v の重みとする．図 6.1(b) では，タスク v_3 において，v_2 の重みと有向辺の重みの和を v_3 の重みとしている．

6.1.2 子孫を持たないタスクからの重み付け

次に，$S_t(G)$ の子孫を持たないタスクからの重み付けについて述べる．図 6.3 を $S_t(G)$ とする．コントロールステップ t では，タスク v が割り当てられる．v の親の集合を $\mathrm{Par}(v)$ とする．$S_t(G)$ からタスク v を除いたグラフ $S_{t-1}(G)$ において，祖先を持たないタスクからの重み付けを行うと，$\mathrm{Par}(v) = \{v_1, v_2, v_3\}$ の重みが $2-\varepsilon$ となり，コントロールステップ $t-1$ で割り当てるタスクが任意に選ばれることになる．2.2 で定義したスケジューリ

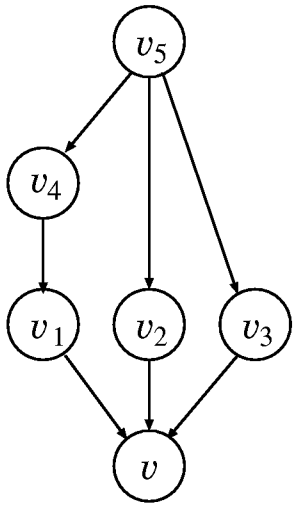


図 6.3: 子孫を持たないタスクからの重み付け例

ングの制約を考慮すれば，v を割り当てたコントロールステップの一つ前のステップ $t-1$ へは，v の親は一つしか割り当てることができない．従って，v の親の一つを $\mathrm{fp}(v)$ として選ぶ．この節で述べる子孫を持たないタスクからの重み付けを行うのは，$\mathrm{fp}(v)$ を決めるためである．子孫を持たないタスクからの重み付けは，最初の入力 G にだけ適用する．また，この重み付けの手順は，6.2 と同様の手順を行うので，細かな説明は省く．図 6.2 に重み付けを行った例を示す．

6.1.3　子孫を持たないタスクから割り当て

DAG G の割り当ては，子孫を持たないタスクの 6.2 で付けられた重みの大きい順に，最終コントロールステップ t_{last} から順に割り当てる．重みが等しいタスクの優先順位は，6.1.3 で決めた重みの小さい方を優先する．

また，6.4(a) のように，あるタスク v が複数個の v の親との間に先行制約関係があり，v がコントロールステップ t に割り当てられている時は，で求まる重みの小さいタスクを $\mathrm{fp}(v)$ とし，$t-1$ へ v と同じ PE に割り当てる．その時，$\mathrm{fp}(v)$ 以外のタスク $\mathrm{Par}(v)\backslash\mathrm{fp}(v)$ は，$t-1$ への割り当てを禁止する．図 6.4(a) は，$\mathrm{fp}(v)$ が v_1 である例を示す．

さらに，複数の親が，共有する子供を複数個持つ時，子供が同じコントロールステップ t に割り当てられた場合，一つ前のコントロールステップ $t-1$ へはどの親も割り当てできない．図 6.4(b) は，v_3, v_4 が v_1, v_2 のどちらも子供に持つグラフの例である．v_1, v_2 が t に割り当てられた時，v_3, v_4 は子供のうち片方の通信遅延は必ず考慮しなければならない．

しかし，このグラフの最適解は，v_1, v_2 を t に割り当てた場合である．よって，この場合は，子供を複数共有する親の割り当てを次の 1 ステップの間強制的に禁止して，その他の割り当て可能なタスクを割り当てできるようにした．

アルゴリズムを図 6.5 に示す．

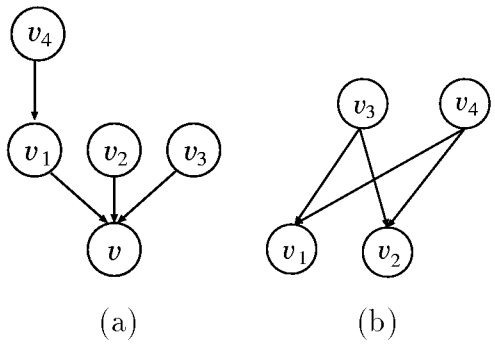


図 6.4: 親を複数持つ例．

6.2 評価

DAG に対する提案手法の評価について述べる．入力となる DAG は，図??のように，タスクを一つ加える毎にそのタスクと先行制約関係を持たせるタスク数とタスクナンバーを，一様乱数により既成のグラフから決定し生成した．実装は，タスク数 $n = 20$ の DAG を 100 サンプル生成し，PE 数を $m = 3$ から $m = 20$ まで行った．

実装の結果，PE 数 $m = 4$ で実装した時に最適解を与えない DAG が見つかった．そのグラフ構造を図 6.7(a) に示す．この DAG を実装した結果，図 6.7(b) に示すように，$m = 4$ のとき v_{12}，v_{14}，v_{17}，v_{20} が最終コントロールステップ割り当てられた．しかし，図 6.7(a) の最適解の一つとして，図 6.7(c) に示す割り当てが考えられる．提案手法が図 6.7(a) のグラフ構造において最適解を与えなかった原因として，以下の理由が考えられる．まず，結論から言うと，最初のステップでの割り当ては v_{14} よりも v_{18} が優先して割り当てられなければならなかった．そのためには，の実行時に，$v_1 1$ の長子として v_{14} を選ばれなければならなかった．しかし，$v_1 1$ の長子として v_{18} が選ばれている．これは，複数の子を持つタスクの長子を決めるアルゴリズムにおいて，子の仮の重み，及び子孫数が等しい場合には，任意に長子を選ぶようになっていることが原因と考えられる．この結果に対する解析，及び改善策は，現在検討している．

program add_weight_for_general_graph G;
integer t= 1;
begin
while($G \neq \emptyset$) {
/* NonSucc(G) からの重み付け */
while(NonSucc(G) から実行する)
{
tmp_w$^{s}_{\mathrm{NonSucc}(G)} = 1$;
各有向辺の重み $w^{s}_{(v,\mathrm{Ch}(v))}$ と
各ノードの仮の重み tmp_w$^{s}_{v}$ を計算する;
}
while(NonSucc(G) から実行する)
{
各ノードの重み w^{s}_{v} を計算する．;
}
/* NonPred(G) 重み付け */
while(NonSucc(G) から実行する)
{
tmp_w$^{p}_{\mathrm{NonPred}(G)} = 1$;
各有向辺の重み $w^{p}_{(v,\mathrm{Ch}(v))}$ と
各ノードの仮の重み tmp_w$^{p}_{v}$ を計算する;
}
while(NonSucc(G) から実行する)
{
各ノードの重み w^{p}_{v} を計算する．;
}
/* PE へ割り当て */
Allocation(G, t);
$t++$;
}
end.
定義
NonSucc(G)：子孫を持たないタスク,
NonPred(G)：先祖を持たないタスク,
w^{s}_{v}：NonSucc(G) からの実行による v の重み,
$w^{s}_{(v_i,v_j)}$：NonSucc(G) からの実行による (v_i, v_j) の重み
tmp_w$^{s}_{v}$：NonSucc(G) からの実行実行による v の仮の重み
w^{p}_{v}：NonPred(G) からの実行による v の重み
$w^{p}_{(v_i,v_j)}$：NonPred(G) からの実行による (v_i, v_j) の重み
tmp_w$^{p}_{v}$：NonPred(G) からの実行による v の仮の重み

図 6.5: 改良手法のアルゴリズム

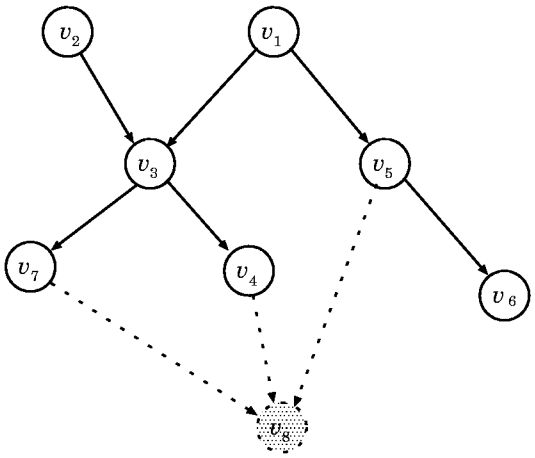
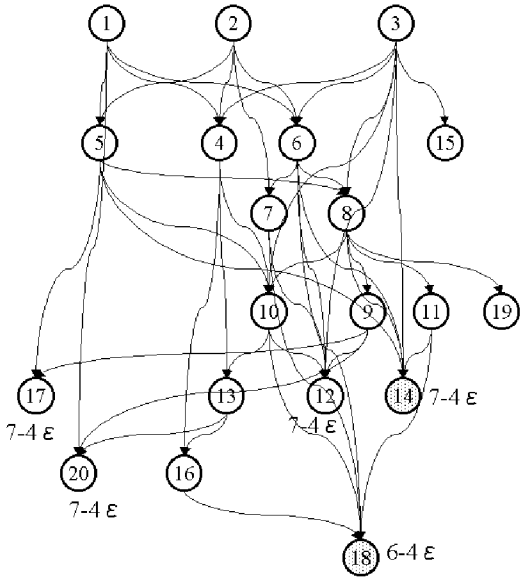


図 6.6: dag の生成 (v_8 が加えられたところ)



(a) 最適解にならない DAG 例.

Control Step	P_1	P_2	P_3	P_4
1	v_3	v_1		
2	v_2			
3	v_6			
4	v_5	v_4		
5	v_8			
6	v_{10}			
7	v_{13}	v_{11}	v_7	
8	v_{16}	v_9		
9	v_{18}	v_{19}	v_{15}	
10	v_{17}	v_{14}	v_{12}	v_{20}

(b) 実装結果.

Control Step	P_1	P_2	P_3	P_4
1	v_3	v_1		
2	v_2			
3	v_6	v_{15}		
4	v_5	v_4		
5	v_8		v_7	
6	v_{10}			
7	v_{13}	v_{11}	v_9	
8	v_{16}	v_{14}	v_{17}	
9	v_{18}	v_{19}	v_{12}	v_{20}
10				

(c) 最適解例.

太線で囲まれたタスクは
同一 PE 上で処理されるタスク.
(タスクに付けられた数値は重みを表す).

図 6.7: 実装結果.

第 7章

まとめ

本研究では，通信遅延を考慮したスケジューリング問題を対象とし，従来の先行制約関係が根付き木で，$W_V(v)$ と $W_A(a)$ が単位時間に制限されている問題に関しての発見的手法を提案した．提案手法では，タスクに優先順位 (重み) を付け，重みの大きいタスクから順に割当を決め，それと同時に重みを付け替える動的な重み付け手法を用いた．各タスクに重みをつけてその重みにより優先順位を決定する提案手法は，従来方法よりも全実行時間は良い結果が得られた．実験による調査の結果，提案手法に関して従来手法より悪い $\mathcal{T}$ の例は，発見されなかった．しかし，重み付けが動的であるため提案手法の近似差に関する解析は困難である．また，根付き木における提案手法を応用して，DAG に対するスケジューリングを求める発見的アルゴリズムを改良手法として提案した．実験の結果，改良手法によって求めた全実行時間が悪くなる DAG の例が見つかった．今後の課題として，動的重み付けを用いた根付き木のスケジューリング手法の解析，及び改良手法の解析を行う．

謝辞

本研究を遂行していく上で随時多大なる御指導，御鞭撻を頂き大変お世話になった北陸先端科学技術大学院大学 金子 峰雄 助教授ならびに田湯 智 助手に深謝致します.

参考文献

[1] T.C.Hu, "Parallel Sequencing and Assembly Line Problems," *Operation Reserch,* vol.9, 1961, pp. 841-848.

[2] Theodora A. Varvatigou, Vwani P. Boychowdhury, Thomas Kailath and Eugene Lawler, "Scheduling In and Out Forests in the Presence of Communication Delays," *IEEE Parallel and Distributed Systems,* vol. 7, No.10, Oct. 1996, pp. 1065-1074.

[3] Michael Pinedo,"Scheduling," Prentice-Hall., 1995.

[4] M.Fujii,T.Kasami,and K.Ninomiya, "Optimalsequencing of two equivalent processors," *SIAM J. Appl.Math.,* vol.17, no.4 1969, pp. 784-789.

[5] H.gabow, "An almost linear algorithm fr two-processor scheduling," *J.ACM,* vol.29, no.3 1982, pp. 766-780.

[6] R. M. Karp, Reducibility among Combinatorial Problems, in Complexity of Computer Computations, R.E. Miller and J.W. Thatchar (eds.), *Plenum Press,* 1972, pp. 85-103.

[7] M.Kaufman, "An almost-optimal algorithm for the assembly line scheduling problem," *IEEE Trans. Comput.,* vol.c-23, no.11 1974, pp. 1169-1174.

[8] C.Papadimitriou and M.Yannakakis, "Scheduling interval-orderd tasks," *SIAM J. Comput.*,vol.8,1979,pp. 405-409.

[9] R. Sethi, "Scheduling graphs on two processors," *SIAM J. Comput.,* vol.5, no.1 1976, pp. 73-82.

[10] J. Ulman, "NP-complete scheduling problems," *J. Comput. System Sci.,* vol.10, 1975, pp. 384–393.