The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks

サマリ

散発的DAGタスクモデル [3]の拡張として，条件付き散発的DAGタスクモデルを提案し，これらのタスクからなるシステムに対するGEDF (Global Earliest-Deadline First)スケジューリングに焦点を当てる．条件付き散発的DAGタスクモデルは，複雑な制御フローを含むタスクをモデル化でき，その表現能力が向上するが，スケジューラビリティテストは複雑になってしまう．そこで，条件付き散発的DAGタスクを多項式時間で(従来の)非条件付き散発的DAGタスクに変換する戦略を提案し，変換したタスクシステムに対して既存のテスト手法 [4]を適用した．結果として，条件付き散発的DAGタスクシステムでも従来の散発的DAGタスクシステムと同じ高速化率であることが示された．

分類

Resource	Properties of tasks	Criterion
P	$dag_i, D_i \leq T_i, prec, pmtn$	-

関連研究との比較

	DAGモデル	条件分岐	疑似多項式時間でのGEDFスケジューラビリティテスト
[7, 12, 1, 10, 5]		-	-
[3, 11, 2, 13, 9]	x		-
[4, 8]	x		x
[6]	x	x	
本研究	x	x	x

- -： 言及なし
- 引用番号は論文内のもの

予備知識

- 非条件付き散発的DAGでは、*volume*(タスク全体の実行にかかる時間コスト)はDAG内のすべてのノードの最悪実行時間(wcet)の和
  - 条件付き散発的DAGではそれほど単純ではない

システムモデル

前提

- 各タスクが条件付き散発的DAG
- ソース・シンクノードは1個ずつ
  - 複数ある場合にはダミーノードを追加することで容易に変換できるため、この仮定は一般性を損なわない
- 分岐係数(条件分岐するノードからのエッジの本数)は2とする
  - 分岐係数が2を超える条件も、DAGのサイズを多項式以上に大きくせずに分岐係数2の条件と等価なシーケンスに変換可能
- ある散発的DAGタスクによって生成されるジョブはすべて共通のリリース時刻とデッドラインを持つ

タスクモデル

Symbols	Descriptions
$\tau_i = (G_i, D_i, T_i)$	条件付き散発的DAGタスク
$G_i = (V_i, E_i)$	DAG
V_i	G_iのノード集合
$\|V_i\|$	G_iのノード数
E_i	G_iのエッジ集合
D_i	τ_iの相対デッドライン
T_i	τ_iの最小リリース間隔
(c_1, c_2)	条件分岐の (開始ノード, 終了ノード)
k	分岐係数; ある分岐開始ノードからの分岐数
$s_1, s_2, \ldots, s_k$	分岐開始ノードc_1の後任ノード
$t_1, t_2, \ldots, t_k$	分岐終了ノードc_2の前任ノード
$V'_\ell \subseteq V_i \ (\ell \in \{1, 2, \ldots, k\})$	s_ℓから到達可能なc_2以外のノード集合
$E'_\ell \subseteq E_i \ (\ell \in \{1, 2, \ldots, k\})$	s_ℓから到達可能なc_2以外のエッジ集合
m	プロセッサ数
n	タスク数
$\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$	タスクシステム
len_i	G_iのクリティカルパス長
$\mathcal{J}_i$	τ_iがとり得るdagジョブの全体集合
$J \in \mathcal{J}_i$	G_iを一度完全に実行して得られる(ノード)ジョブ列
vol_i	すべてのJにおける、J内の全ジョブの合計wcetの最大値
$\delta_i = \mathrm{len}_i / D_i$	τ_iの密度

Symbols	Descriptions
$U_i = \mathrm{vol}_i / T_i$	τ_iの利用率
$\delta_{\max}(\tau) = \max_{\tau_i \in \tau}\{\delta_i\}$	τの最大密度
$U(\tau) = \sum_{\tau_i \in \tau} U_i$	τの総利用率
s	プロセッサの速度

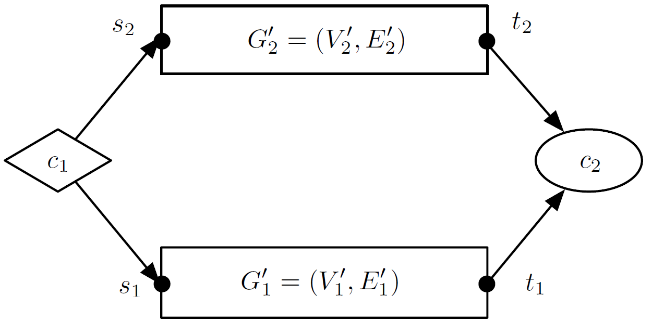

Fig. 1. A canonical conditional construct with branching factor 2. Vertices s_1 and t_1 (vertices s_2 and t_2, resp.) are the sole source vertex and sink vertex of G'_1 (G'_2, resp.).

- G_iの条件分岐ノードの各対(c_1, c_2)に対して，c_1で始まりc_2で終わるの部分グラフを，G_iの**条件付き構成**と呼ぶ
- 条件付き構成の中には，さらに条件付き構成があっても良い
    - 中に条件付き構成を含まないような条件付き構成を，**最も内側の条件付き構成**という用語で説明する

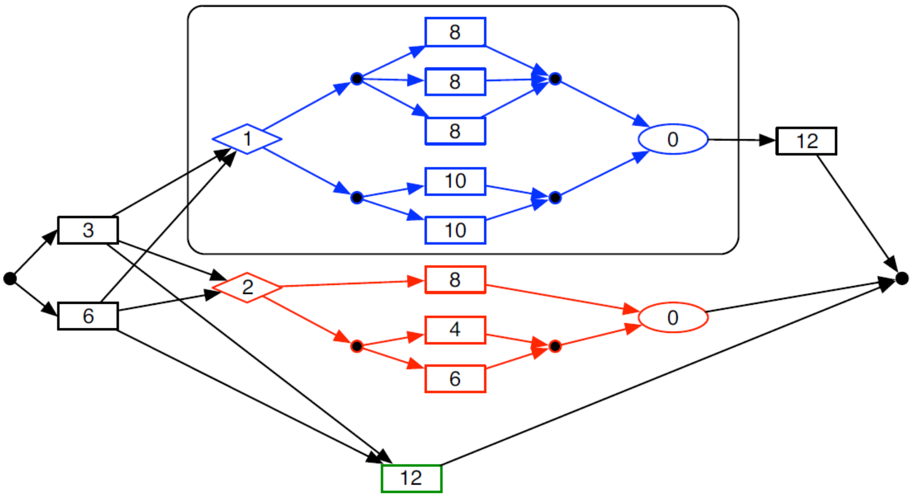

Fig. 2. The DAG of an example conditional sporadic DAG task. Vertices denote jobs; the numbers within vertices denote the wcets of the jobs. Small solid circles denote jobs with wcet equal to zero. Diamonds and ovals denote conditional start and end vertices respectively, and rectangles denote non-conditional vertices. (The large rectangle encloses a single conditional construct in the DAG that will be referenced later in this document.)

- $\mathrm{len}_1 = (6 + 1 + 10 + 0 + 12) = 29$
- $\mathrm{vol}_1 = 6 + 3 + 25 + 12 + 12 + 12 = 70$

- 上の条件付き構成で上の経路($1+(8\times 3)+0=25$)が選択され，下の条件付き構成で下の経路($2+(4+6)+0=12$)が実行される場合に合計wcetが最大

アプローチ

非条件付き散発的DAGタスクシステムGEDFスケジューリング(先行研究)

いま，与えられた散発的DAGタスクシステムτの実行に利用可能な速度sのプロセッサが無限個あるとし，τによって生成されたジョブ集合をJとする．$S_\infty(J,s)$は，Jの各ジョブが実行可能になった瞬間に速度sのプロセッサを割り当て，割り当てられたプロセッサでジョブの実行が完了するまで実行することで得られるスケジュールを表すとする．

Definition 1 (*work*関数)：
τ_iを散発的DAGタスク，$s\leq 1$を正の実数，τ_iによって合法的に生成されるジョブの集合をJとする．
任意の区間Iに対して，$\mathrm{work}(J,I,s)$は，スケジュール$S_\infty(J,s)$において，区間I内に実行され，デッドラインを持つジョブの実行量を表す．
任意の正の整数tに対して，$\mathrm{work}(J,t,s)$は，長さtの任意の区間Iにおいて，$\mathrm{work}(J,I,s)$が取り得る最大値を表す．
最後に，$\mathrm{work}(\tau_i,t,s)$は，散発的DAGタスク$\tau_i$によって生成される全てのジョブ列$J$に対して，$\mathrm{work}(J,t,s)$の最大値を表す．

つまり，$\mathrm{work}(\tau_i,t,s)$は，$\tau_i$によって生成されうるすべてのジョブ列$J$の中で，スケジュール$S_\infty(J,s)$において，長さ$t$の区間内に実行され，**この区間内にデッドラインを持つ**Jのジョブの実行量の最大値として定義される。したがって，τ_iの全てのデッドラインを常に満たすことができるスケジュールは、τ_iのジョブを長さtの区間で少なくとも$\mathrm{work}(\tau_i,t,s)$だけ実行できなければならない．

Theorem 1 ([4])：
散発的DAGタスクシステムτは，任意の$t\geq 0$に対して次式を満たすような定数$\sigma,\delta_{\max}(\tau)\leq\sigma\leq 1$が存在するときに$m$個の単速度プロセッサでGEDFスケジュール可能．

$$\sum_{\tau_i\in\tau}\mathrm{work}(\tau_i,t,\sigma)\leq(m-(m-1)\sigma)\times t \tag{1}$$

したがって，与えられたτがm個の単速度プロセッサ上でEDFスケジューリング可能であることを示すには，上記のTheorem 1に従い，すべての$t\geq 0$に対して条件式(1)が成立するようなσの値を得れば十分．[4]のスケジューラビリティテストは，本質的に$\sigma\leftarrow m/(2m-1)$がそのような値であるかを決定することに帰着する．この事実が散発的DAGタスクシステムのGEDFスケジューリングに$(2-1/m)$の高速化境界を与えることが[4]で示された．

Corollary 1 ([4])：
GEDFは，散発的DAGタスクシステムをm個のプリエンプティブプロセッサでスケジュールする場合，高速化率が$\left(2-\frac{1}{m}\right)$となる．

擬似多項式時間でのスケジューラビリティテスト：
[4]のスケジューラビリティテストは，本質的に$\sigma \leftarrow m/(2m-1)$が全ての$t \geq 0$に対して条件式(1)を成立させるかを決定することからなり，複数のtに対して$\mathrm{work}(\tau, t, m/(2m-1))$を計算する必要がある．[4]では，これを擬似多項式時間で効率的に実行できることが示され，結果として得られる擬似多項式時間GEDFスケジューラビリティテストは，任意の定数$\epsilon > 0$に対して高速化率$(2-1/m+\epsilon)$となる．

条件付き散発的DAGタスクシステムのGEDFスケジューリング

*work*関数の定義を条件付き散発的DAGタスクに拡張し，Theorem 1とCorollary 1が条件付き散発的DAGタスクシステムに対して依然成立することを示す．

条件付き散発的DAGタスクの*work*関数：
Jについて，ジョブの「合法的な」集合という概念が，従来の散発的DAGタスクではなく，条件付き散発的DAGタスクでの意味で満足されなければならないことを理解したうえで，Definition 1全体は変更を加えることなく条件付き散発的DAGタスクに拡張される．

Theorem 1とCorollary 1の拡張：
[4]のTheorem 1とCorollary 1の証明では，与えられたタスクシステムによって合法的に生成されうるすべてのジョブの集合に対する最悪の場合の振る舞いを考慮することに基づいている．これらの証明で必要とされるジョブ集合の唯一の性質は，一対のジョブ間に優先度制約がある場合，これらのジョブ同じリリース時刻とデッドラインを持つことのみ．この性質は，条件付き散発的DAGタスクシステムによって生成されたジョブ集合である以上，自明に満たされる．したがって，証明は変更されず，Theorem 1とCorollary 1の条件付き散発的DAGタスクへの拡張として，以下のTheorem 2とCorollary 2が直ちに結論づけられる．

Theorem 2：
条件付き散発的DAGタスクシステムτは，任意の$t \geq 0$に対して次式を満たすような定数$\sigma, \delta_{\max}(\tau) \leq \sigma \leq 1$が存在するときに$m$個の単速度プロセッサでGEDFスケジュール可能．

$$\sum_{\tau_i \in \tau} \mathrm{work}(\tau_i, t, \sigma) \leq (m-(m-1)\sigma) \times t$$

Corollary 2：
GEDFは，条件付き散発的DAGタスクシステムをm個のプリエンプティブプロセッサでスケジュールする場合，高速化率が$\left(2-\frac{1}{m}\right)$となる．

条件付き散発的DAGタスクシステムのGEDFスケジューラビリティテスト

*work*関数の導出

Corollary 2は，高速化率の観点から，条件付き構成の表現を可能にするために散発的DAGタスクモデルを拡張しても，追加のコストやペナルティが発生しないことを立証する．しかし，この結果自体は，条件付き散発的DAGタスクシステムのGEDFスケジューラビリティテスト手法を与えない．以下で，条件付き散発的DAGタスクシステムに対する効率的な(擬似多項式時間の)GEDFスケジューラビリティテストを導出する．

Definition 2 (*rdem*関数 $\mathrm{rdem}(\tau_i, t, s)$)：
与えられた条件付き散発的DAGタスクτ_iの任意の$J \in \mathcal{J}_i$を考え，$t \le D_i$を任意の正の実数とする．$\mathrm{rdem}(J, t, s)$は，スケジュール$S_\infty(J, s)$において，Jのジョブのリリース時刻からt時間単位後の残りの仕事量 — すなわち，Jに含まれる全ジョブの合計wcetから，既に実行されたものを引いたものを表す．$\mathrm{rdem}(\tau_i, t, s)$は，全ジョブ集合$J \in \mathcal{J}_i$における$\mathrm{rdem}(J, t, s)$の最大値と定義される．

Lemma 1：
τ_iを制約付きデッドライン-条件付き散発的DAGタスクとする．すべての$s \ge \delta_i$および$t \le D_i$に対して，次式が成り立つ．

$$\mathrm{work}(\tau_i, t, s) = \mathrm{rdem}(\tau_i, D_i - t, s) \tag{2}$$

$\mathrm{work}(\tau_i, t, s)$の値を定義するシナリオは，長さ$t$の区間の右端とデッドラインが一致する$\tau_i$のdagジョブがあり，$\tau_i$の他のdagジョブが可能な限り接近してリリースされるものである [4]．τ_iの完全なdagジョブは，この区間内に$\lfloor t/T_i \rfloor$個存在し(Fig. 3)，これらのdagジョブの最大の仕事量はvol_iである．残りの区間(長さ$t \bmod T_i$)内にデッドラインを持つジョブの実行量を決定する必要がある．$(t \bmod T_i) \ge D_i$であれば，dagジョブ全体がこの区間に収まるため，その仕事量はvol_iであり，次の関係が成り立つ．

$$\mathrm{work}(\tau_i, t, s) = \mathrm{vol}_i \times \lfloor t/T_i \rfloor + \begin{cases} \mathrm{vol}_i, & \text{if } (t \bmod T_i) \ge D_i \\ \mathrm{work}(\tau_i, t \bmod T_i, s), & \text{if } (t \bmod T_i) \le D_i \end{cases}$$

$(t \bmod T_i) \le D_i$の場合にLemma 1を適用して，次式を得る．

$$\mathrm{work}(\tau_i, t, s) = \mathrm{vol}_i \times \lfloor t/T_i \rfloor + \begin{cases} \mathrm{vol}_i, & \text{if } (t \bmod T_i) \ge D_i \\ \mathrm{rdem}(\tau_i, D_i - (t \bmod T_i), s), & \text{if } (t \bmod T_i) \le D_i \end{cases} \tag{3}$$

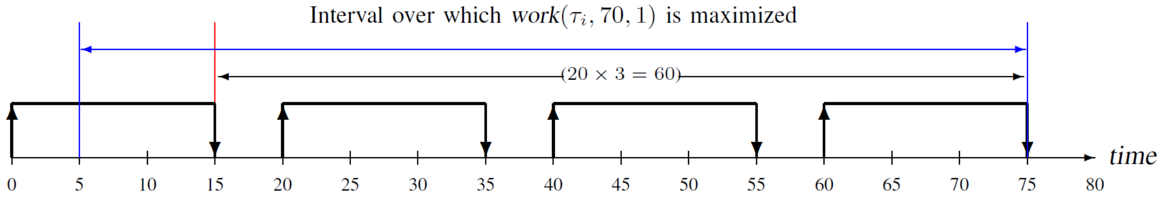

Fig. 3. Computing $work(\tau_i, 70, 1)$ for the task τ_i of Figure 4.

$D_i = 15, T_i = 20, t \leftarrow 70$のタスクの例

これにより，すべてのtについて$\mathrm{work}(\tau_i, t, s)$を計算する問題を，$t \le D_i$の値について$\mathrm{rdem}(\tau_i, t, s)$を計算する問題に縮小した．残る問題は，$t \le D_i$の値に対して$\mathrm{rdem}(\tau_i, t, s)$をどのように計算するかのみ．

まず，単位速度プロセッサのプラットフォームで実行されるタスクτ_iの$\mathrm{rdem}(\tau_i, t, 1)$を計算する例を挙げる．このタスクは，$D_i = 15$および$T_i = 20$であり，Fig. 4に示すようなDAG G_iを持つ．G_iでは，τ_iのdagジョブがリリースされるたびに，$\mathrm{wcet} = 1$を持つ条件式が評価される．この評価の結果に応じて，それぞれ並列実行可能な$\mathrm{wcet}{=}8$の3つのジョブか，$\mathrm{wcet}{=}10$の2個のジョブが実行される．分岐の再合流にかかる実行コストはないため，条件分岐の終了ノードは$\mathrm{wcet}{=}0$．

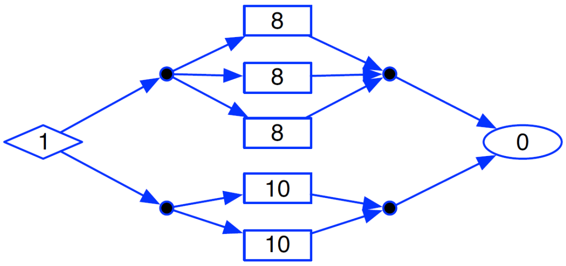

Fig. 4. The DAG G_i of an example conditional DAG task τ_i. For this task, $D_i = 15$ and $T_i = 20$. Note that $len_i = 11$ and corresponds to taking the lower branch of the conditional, while $vol_i = 25$ and corresponds to taking the upper branch.

Fig. 2の四角形で囲まれた部分の条件付き構成となっている

この例のタスクτ_iについて，tの関数として$\mathrm{rdem}(\tau_i, t, 1)$を決定する．このタスクでは$|\mathcal{J}_i| = 2$である．すなわち，このタスクの1個のdagジョブに対して，このDAGを通る制御フローは2通り考えられる(上の経路 or 下の経路)。ここでは，上の経路と下の経路を別々に考え，その結果の関数をFig. 5にグラフで示す．

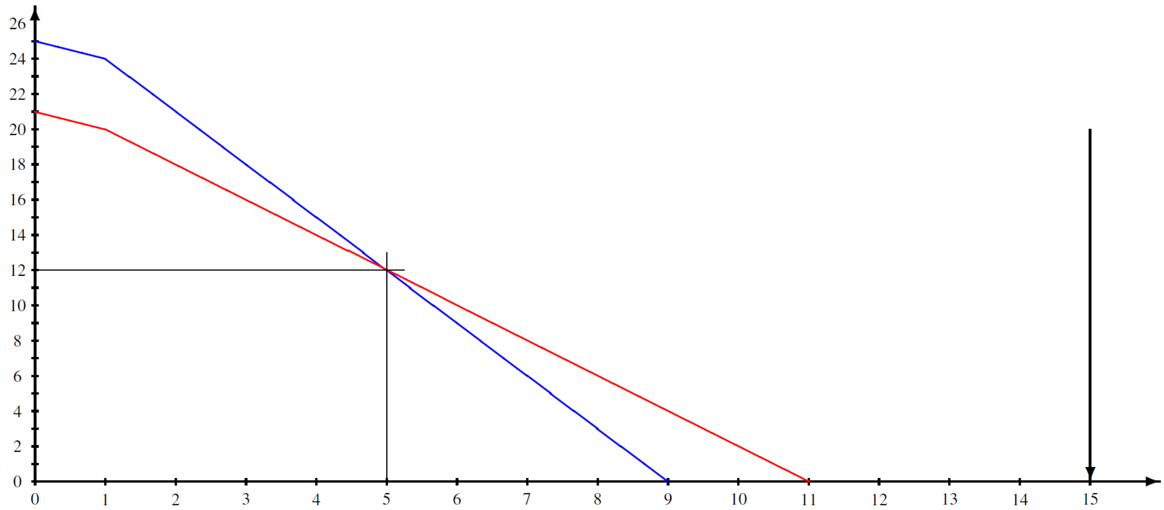

Fig. 5. Remaining work (y axis) as a function of time elapsed (x axis) since the release of a dag-job of the task τ_i.

- **上の経路を通る場合 (青線)．**残余仕事量は、点$(0, 25)$を始点とし，最初の時点で25単位の仕事が残っている．区間$[0, 1)$で実行される仕事は条件式のノードに対応する1単位のみであるため，この区間の直線の傾きは-1．条件式が評価されると，3個のジョブが8時間単位で並行して実行されるため，傾きは-3となる．
- **下の経路を通る場合 (赤線)．**残余仕事量は，点$(0, 21)$を始点とし，最初の時点で21単位の仕事が残っている．上の場合と同様に，条件付きノードに対応する仕事だけが区間$[0, 1)$で実行され，この区間の線の傾きは-1．条件式が評価されると，2個のジョブが10時間単位で並行して実行されるため，傾きは-2となる．

図から明らかなように，$t \leq 5$の場合には上の経路を実行するとt時間単位後に多くの仕事が残り(青線)，$t \geq 5$の場合には下の経路を実行するとより多くの仕事が残る(赤線)．つまり，二つの異なる経路に対応する個々の*rdem*関数の上側包絡線は，すべてのtの値に対する残余仕事量の最大

値を表す．したがって，$\mathrm{rdem}(\tau_i, t, 1)$は，Fig. 5にプロットされた二つの個別の*rdem*線の上側包絡線となる．

次に式(3)を適用して，Fig. 4のタスクの例について，$t = 65, 70, 72, 78$の場合の$\mathrm{work}(\tau_i, t, 1)$を計算する例を挙げる．すべての$t$について，$\lfloor t/T_i \rfloor = 3$かつ$\mathrm{vol}_i \times \lfloor t/T_i \rfloor = 25 \times 3 = 75$となっている．$(t \bmod T_i)$は，$t = 65, 70, 72$では$\leq D_i$だが，$t = 78$では$> D_i$となる．各$t$の値に対して式(3)を別々に適用することにより、以下を得る．

$$
\begin{aligned}
&\mathrm{work}(\tau_i, 65, 1) = 75 + \mathrm{rdem}(\tau_i, 10, 1) = 75 + 2 = 77 \\
&\mathrm{work}(\tau_i, 70, 1) = 75 + \mathrm{rdem}(\tau_i, 5, 1) = 75 + 12 = 87 \\
&\mathrm{work}(\tau_i, 72, 1) = 75 + \mathrm{rdem}(\tau_i, 3, 1) = 75 + 18 = 93 \\
\text{and} \quad &\mathrm{work}(\tau_i, 78, 1) = 75 + \mathrm{vol}_i = 75 + 25 = 100
\end{aligned}
$$

Fig. 5で例として示した*rdem*関数を計算するアプローチは，制約付きデッドライン-条件付き散発的DAGタスクτ_iの*rdem*関数を計算するための簡単な方法論として容易に一般化可能：

- 各$J \in \mathcal{J}_i$について$\mathrm{rdem}(J, t, s)$を計算する
- 得られた個々の$\mathrm{rdem}(J, t, s)$の上側包絡線が$\mathrm{rdem}(\tau_i, t, s)$となる

条件付き散発的DAGタスクから非条件付き散発的DAGタスクへの変換

ここまでのアプローチは正しいが，[6]と同様の問題を抱える：
$|\mathcal{J}_i|$はDAGのサイズに対して指数関数的になりうるため，アルゴリズム全体は指数関数的な時間を要する．そこで，この方法で*rdem*関数を明示的に計算する代わりに，*rdem*関数の特性に関して上記で得た洞察を利用して、条件付き散発的DAGタスクシステムのGEDFスケジューラビリティ分析のより効率的なアプローチを開発する．与えられたタスクシステムがTheorem 2を満たすかどうかを直接判断するのではなく(そのためには*work*関数を明示的に計算する必要がある)，代わりに各条件付き散発的DAGタスクτ_iを非条件付きタスクτ_i'に効率的に**変換**する．

変換前後の二つのタスクは，同一の*len*，*vol*，デッドライン，周期を持つという意味で「等価」であり，すべての$t \geq 0$および$s \geq \delta_i$に対して，次が成り立つ．

$$\mathrm{work}(\tau_i, t, s) = \mathrm{work}(\tau_i', t, s) \tag{4}$$

したがって，条件付き散発的DAGタスクシステムτは、τの各タスク変換して得られる非条件付き散発的DAGタスクシステムτ'がTheorem 1を満たす場合にのみ，Theorem 2を満たす．そして、[4]の非条件付き散発的DAGタスクシステムに対する擬似多項式時間GEDFスケジューラビリティテストを非条件付き散発的DAGタスクシステムτ'に適用することで，任意の定数$\epsilon > 0$に対して高速化率$(2 - 1/m + \epsilon)$の条件付き散発的DAGタスクシステムに対する擬似多項式時間GEDFスケジューラビリティテストを得られる．

Fig. 4の例のタスクを変換する例を挙げる．$D_j = D_i = 15, T_j = T_i = 20$であり，Fig. 6に示すDAG G_jを持つタスクτ_jを考える．このタスクの*rdem*関数は、Fig. 5に描かれた二つの*rdem*関数の上側包絡線と同一であることが容易に検証可能．したがって，**タスクτ_iとτ_jは同一の*rdem*関数(**すなわち同一の*work*関数)を持ち，τ_jは，同一の*work*関数を持つという意味でτ_iと「等価」な非条件付き散発的DAGタスクである．

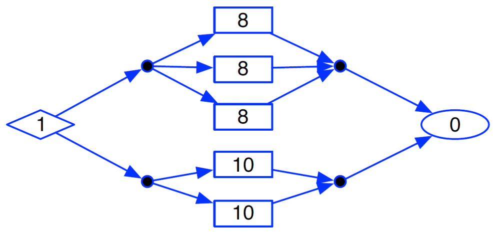

Fig. 4. The DAG G_i of an example conditional DAG task τ_i. For this task, $D_i = 15$ and $T_i = 20$. Note that $len_i = 11$ and corresponds to taking the lower branch of the conditional, while $vol_i = 25$ and corresponds to taking the upper branch.

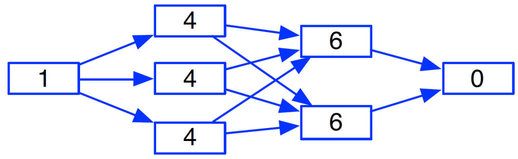

Fig. 6. The DAG G_j of an example conditional DAG task τ_j.

τ_jを得る方法：

基本的にFig. 5を検証し，二つの*rdem*関数の上側包絡線に等しい*rdem*関数を持つ非条件付き散発的DAGタスクの構築を目指した．

- 上側包絡線は区間$[0, 1)$で-1の傾きを持つため，$\mathrm{wect} = (1 - 0) = 1$のノードを1個導入した．
- 上側包絡線の傾きは区間$[1, 5)$で-3である．これは，$\mathrm{wect} = (5 - 1) = 4$の3個のノードからなる2番目の「層」を後任ノードとして追加することでモデル化される．
- 上側包絡線の傾きは区間$[5, 11)$で-2である．これは，$\mathrm{wect} = (11 - 5) = 6$の2個のノードからなる3番目の層を後任ノードとして追加することでモデル化される．
- $\mathrm{wect} = 0$の1個のノードを持つ最後の層は，条件付き構成の終わりを表す．
- (各層の各ノードから直後の層のすべてのノードへのエッジを追加)

ただし，個々の*rdem*関数の上側包絡線を得るためには，まず，すべての(おそらく指数関数的に多数の)ありうる制御フローについて，これらの個々の*rdem*関数を明示的に計算しなければならないため，これだけではこの手法が以前の手法より効率的には見えない．条件付き散発的DAGタスクを等価な非条件付きタスクに効率的に変換するためには，もう一つの成果が必要となる．以下に例を挙げてその成果を説明する．

効率的な変換アルゴリズム

Fig. 4のDAGは，Fig. 2のDAGの中の一つの条件付き構成として現れる．もし，Fig. 2のDAGの条件付き構成全体をFig. 6のDAGに置き換えると，**条件付き構成が1つ少ない**条件付きDAGが得られ，(以下のTheorem 3により)*work*関数はFig. 2のDAGの*work*関数と同じになる．Fig. 2のDAGのもう一つの(下の)条件付き構成についても同様に考えられる．Fig. 7は，この下の条件付き構成に同様の変換を適用したものである．Fig. 8は，両方の変換を適用した結果の非条件付きDAGを示している．

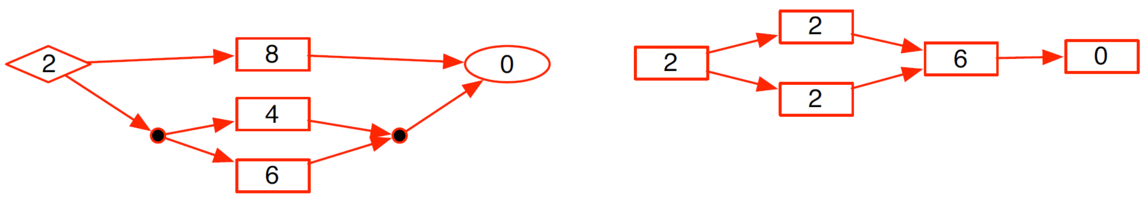

Fig. 7. Transforming the lower conditional construct of the DAG of Figure 2.

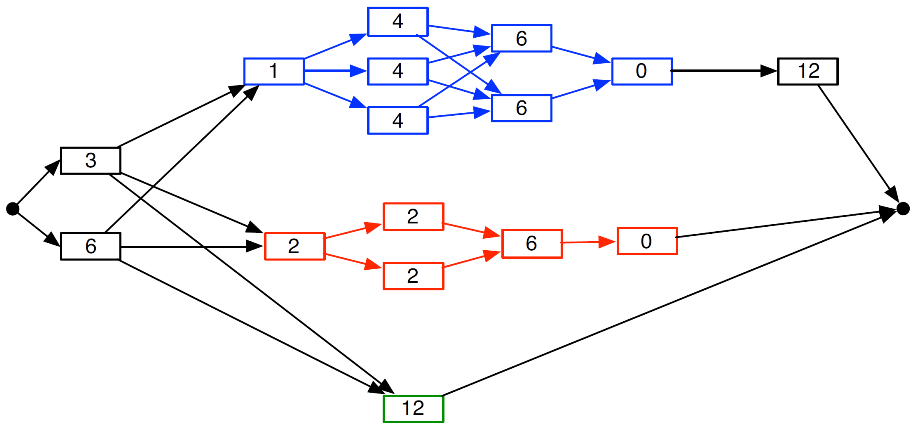

Fig. 8. The DAG of Figure 2 with both conditional constructs removed.

Fig. 2の条件付きDAGに対して，上の条件付き構成をFig. 4のDAGで，下の条件付き構成をFig. 7のDAGで置き換えて得られた非条件付きDAG

変換アルゴリズム：

条件付き散発的DAGタスクτ_iを等価な非条件付き散発的DAGタスクτ_i'に変換する．$D_i' \leftarrow D_i$および$T_i' \leftarrow T_i$である．G_iから開始し，DAG内に条件付き構成がなくなるまで以下を繰り返してG_i'を得る．

1. 最も内側の条件付き構成を特定する
2. 特定した条件付き構成と等価な非条件付きDAGを構築する(方法は後述)
3. 特定した条件付き構成を、等価な非条件付きDAGに置き換える

等価な非条件付きDAGの構成：

Fig. 1に表されている最も内側の条件付き構成に相当する非条件付きDAGを求めることを考える．

- $\{c_1, c_2\} \cup V_1'$と$\{c_1, c_2\} \cup V_2'$のすべてのノードにそれぞれ対応するジョブ集合の*rdem*関数を別々に構築する．各*rdem*は区分線形であり，区分数はグラフのノード数によって上界され，各線分は負の整数の傾きを持つ．
- これらの二つの*rdem*関数の上側包絡線を決定する．この上側包絡線も区分線形となり，各線分は負の整数の傾きを持ち、区分の総数は$\{c_1, c_2\} \cup V_1' \cup V_2'$ のノード数で上界される．
- 上で決定した上側包絡線と同じ*rdem*関数を持つDAG $G'' = (V'', E'')$を構築する．このグラフは，$(\text{上側包絡線の区分数}) + 1$層からなる「層状」グラフとして構成される．第k層のノード数は上側包絡線の第k区分の(負の)傾きに等しく，各ノードは第k区分の時間軸の長さに等しいwcetを持つ．最後の層は$\text{wcet} = 0$の1個のシンクノードからなる．各ノードから直後の層の各ノードへのエッジが存在する．

このDAG $G'' = (V'', E'')$は等価な非条件付きDAGである．

Theorem 3：

τ_iは制約付きデッドライン-条件付き散発的DAGタスクを表し，$\hat{\tau}_i$はτ_iのDAG G_iの最も内側の条件付き構成を等価な非条件付きDAGで置き換えることによって得られる(条件付き or 非条件付きの)散発的DAGタスクを表すとする．

すべての$t, 0 \le t \le D_i$について，$\mathsf{redem}(\tau_i, t, s) = \mathsf{redem}(\hat{\tau}_i, t, s)$
(したがって，すべてのtについて$\mathsf{work}(\tau_i, t, s) = \mathsf{work}(\hat{\tau}_i, t, s)$).

条件付き散発的DAGタスクのvol_iの算出

条件付き散発的DAGタスクを上記の変換によって等価な非条件付き散発的DAGタスクに変換し，変換後のDAGの全ノードのwcetの単純和としてvol_iが得られる．vol_iが得られれば，残りのパラメータ(各タスクの$\mathrm{len}_i, \delta_i, U_i$およびタスクシステムの$\delta_{\max}(\tau), U(\tau)$)は非常に効率的に導出できる．