成蹊大学理工学研究報告
J. Fac. Sci. Tech., Seikei Univ.
Vol.45 No. 1 (2008) pp.5-12

タスクスケジューリングのための GUI ツールの開発

色川　恵理*[1]，石岡　秀基*[2]，甲斐　宗徳*[3]

Development of a GUI tool supporting Task Scheduling

Eri IROKAWA*[1], Hideki ISHIOKA*[2], Munenori KAI*[3],

ABSTRACT : In this paper, we propose a very useful GUI tool to construct some algorithms to solve any task scheduling problems. In the past works in order to construct algorithms to solve task scheduling problems, we had to draw task graphs to know their parallelism structure, and draw Gantt-charts to analyze and adjust the scheduling results. The GUI tool for task scheduling which we designed and implemented can visualize both task graphs and Gantt-charts automatically, and make them cooperate to evaluate and reconstruct task scheduling algorithms.

Keywords：task scheduling, GUI tool, task graph, Gantt chart

（Received March 25, 2008）

１．はじめに

近年のプロセッサは，大量の計算処理を高速に行うために改良されているが，単一プロセッサの処理能力の向上には限度があるといわれている。しかし，コンピュータを使用するアプリケーションは，より大規模化・複雑化し，計算量が増えつつある。そこで，処理を高速化するために，プロセッサを複数台使用して一つの巨大な計算機のように動くマルチプロセッサシステムの研究がハードウエア，ソフトウェアの両面で進められている。このような，マルチプロセッサシステムで適切な負荷分散を行い，処理の高速化を実現するためには，プログラムを解析し並列実行可能なタスク集合を見つけ，個々のタスクを適切なプロセッサに割り当てるためのスケジューリング手法が，重要なキーポイントになる。しかし，このスケジューリング問題は，強 NP 困難なクラスに属す組み合わせ最適化問題であり，大規模タスクグラフにおいて，最適解を求めることは事実上困難である。そのため，実用的な時間内で，最適または近似解を求めるスケジューリング手法が提案されているが，これは，人間の経験に基づく，ヒューリスティックアルゴリズムであるため，得られた解が最適解であるという保証はない。また，全探索アルゴリズムでは，理論的には最適解が求まるが，実際には，実用的な時間で最適解を得ることはほとんど不可能である。そこで，プログラムを解析し，並列実行可能なタスク集合を見つけて，そのタスク集合のプロセッサへの割り当て方を考察する，すなわち，探索にヒューリスティックを組み合わせたスケジューリング手法というものが１つのキーポイントとなる。

このようにスケジューリングの良し悪しが並列処理の効率を左右する一因となるが，そのアルゴリズムを構築するためにはタスクグラフの形状や暫定解の見直しによるヒューリスティックの考察が必要である。従来はタスクグラフやガントチャートを人手により作成して考察していたが，多数のタスク集合を材料にしてスケジューリングアルゴリズムを練り上げるために，大いに手間と時間がかかる状況にあった。そこで本研究では，スケジューリングアルゴリズムや必要なヒューリスティックを考察する手助けとするためにタスクグラフとガントチャートを自動で表示し，さらにスケジューリングアルゴリズムの改善に役に立つ GUI ツールの開発を行った。

２．タスクグラフとガントチャート

2．1　タスクグラフ

１つのプログラムはタスクの集合である。タスク集合は，タスクをノード，先行制約を有向エッジで表した無サイクル有向グラフで表現できる。このグラフをタスクグラフと呼ぶ。図 2.1 がタスクグラフの図で，ノードの

*[1]：工学研究科情報処理専攻修士学生
*[2]：工学部経営・情報工学科学生
*[3]：工学研究科情報処理専攻教授 (kai@is.seikei.ac.jp)

中の数字がタスク番号，ノード脇の数字がタスクの実行時間を示している。

タスクスケジューリング問題とは，このようなタスクグラフの各タスクを与えられた個数のプロセッサに並列処理時間を最小にすることを目的にして割り当てを決定することである。

2. 2 ガントチャート

ガントチャートとは，タスクスケジューリング結果，すなわちタスクのプロセッサへの割り当て結果を図示したものであり，どのタスクをどのタイミングでどのプロセッサに割り当てるかを可視化したものである。図 2.2 のガントチャートは，図 2.1 のタスクグラフのスケジューリング結果の例を図示したものである。

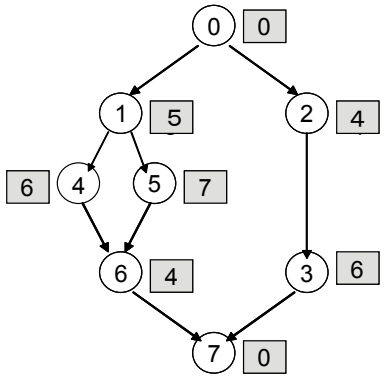

図 2.1　タスクグラフ

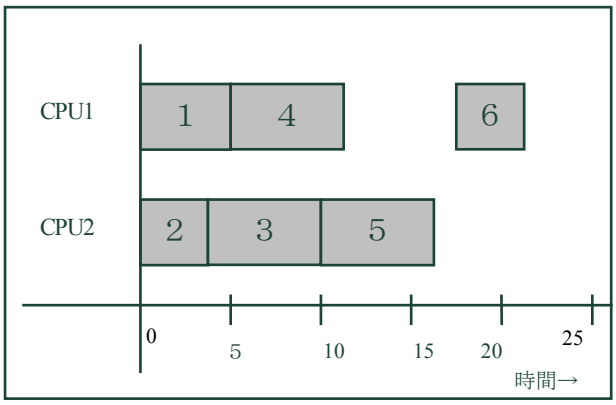

図 2.2　ガントチャート

3．タスクスケジューリング用 GUI ツール[2][3]

これまで，スケジューリングを行う上で問題になっていたことが二つある。一つは探索により最適解を見つけるまでに時間がかかってしまう事である。もう一つは実用的な時間内でスケジューリング結果を求めた際にその解が最適解にどれだけ近いのかを評価したり解の改善の余地をスケジューリング結果から見出したりすることが難しいということである。

スケジューリング結果はどのタスクが，どのプロセッサにどの時刻で割り当てられているかという情報であり，例えば図 3.1 のような数字の羅列で表現されている。これを見ているだけでは図 2.1 や図 2.2 のような図のイメージをつかむことは難しい。以前は図 3.1 のようなスケジューリング結果を元に人手でタスクグラフやガントチャートを作成し，得られた並列処理スケジュールの良し悪しを考察していたので作業効率が低かった。そこで，タスクグラフやガントチャートを表示するツールで図 3.1 のような結果情報を読み込み，自動的に視覚化することで手間をかけずに正確なグラフを確認することが出来るようになる。

```
タスクグラフ
7
0 2 0
1 6 1 0
2 3 1 0
3 10 1 1
4 10 1 1
5 12 2 1 2
6 10 3 3 4 5

0 1 0
```

```
ガントチャート
2

0 1 3 5 6
2 4

0 2 8 19 31
4 14

2 5 7 12
4 6 24 6
```

図 3.1　ツール用データフォーマット(一部)

また，タスクに関する情報を表示する機能を追加することで，通信や先行タスクの確認などが可能になる。さらにタスクを移動する機能や部分的スケジューリング機能を利用し，手動でタスクを操作してスケジューリングを調整することでさらに良い解が見つかる可能性がある。既存のアルゴリズムのみでは発見できなかったが，人による操作や考察を加えてさらに良い解を発見しその手法をアルゴリズムに反映させる，などの新たなヒューリスティック解法の発見の助けにもなる。

3. 1　ツール概観

図 3.2 に示すように，タスクグラフとガントチャートを同時に参照するために複数ウィンドウを起動させ，それぞれにタスクグラフとガントチャートを表示させるマルチウィンドウ方式で作成している。これにより，ガントチャートとタスクグラフが一括して表示できるだけでなく，それぞれを連携させた機能を持たせることも可能になる。

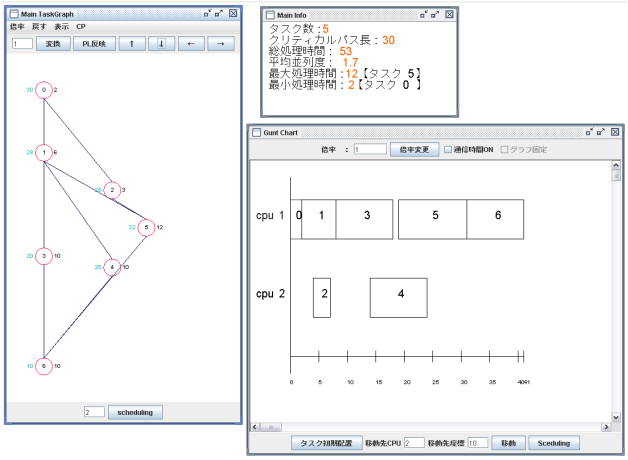

図 3.2　補助ツール概観

3. 2　タスクグラフツールの基本機能

①　拡大，縮小機能

タスクグラフ上部にある入力フォームに数字を入力すると，その値の倍率でタスクグラフを再描画する。また，メニューバーにも倍率変更メニューが存在し，そちらからでも倍率変更が可能となっている（図 3.3 参照）。この機能で倍率を調整し，タスクグラフの必要な部分を拡大したり全体表示するために縮小したりなどの調整ができる。

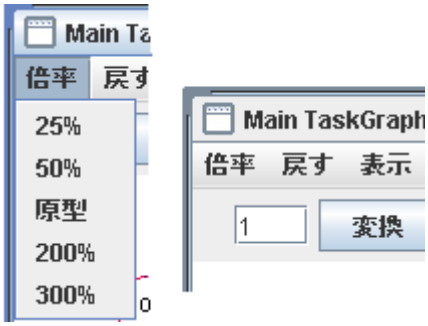

図 3.3　タスクグラフ倍率変更

②　タスクの適切な配置機能

図 3.1 のようなタスクグラフデータに基づく初期描画では，タスクを表すノードの位置は，先行関係を考慮しながら縦軸方向は上からプライオリティレベルの高い順に，横軸方向はほぼ等しいプライオリティレベルを持つタスクを一定間隔で配置する。プライオリティレベルとは，あるタスクから出口ノードが終了するまでの処理時間の下限値である。入口ノードのプライオリティレベルは並列実行時間の下限値になっているが，通信時間は考慮されていない。通信を考慮するとクリティカルパス長に事実上通信時間が影響を及ぼす場合があり，その分だけ本当のプライオリティレベルは大きくなる。しかし，通信を考慮したプライオリティレベルを求めるにはスケジューリングと同様に探索問題を解かねばならないため，ここでは通信時間を無視している。このようなプライオリティレベルを，ノードを表示する位置に反映することで大体同じ時期に並列して処理を行うべきタスクを確認することが出来る。

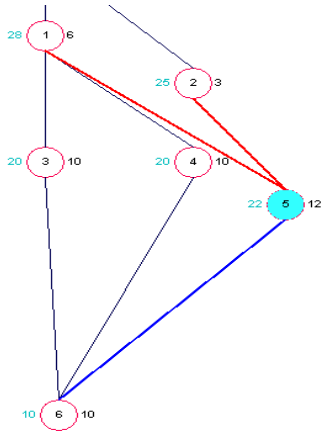

図3.4　タスク５選択(グラフ一部)

タスク数が増加すると，タスク間のエッジが重なってしまうなどして，見難くなってしまうことがある。この時，グラフを見やすくするためにタスクをドラッグ＆ドロップして任意の場所に移動することができる。そのときタスクの先行エッジが赤い色に，後続エッジが青い色に変化する（図 3.4 参照）。ただし手動による移動では，プライオリティレベルを反映した位置に必ずしも配置できない場合が起こり得る。例えば図 3.4 のようにタスク 5 のプライオリティレベルが 22 だが，手動でタスクを動かした為にプライオリティレベル 20 のタスク 3 や 4 よりも低い位置になってしまったとする。このようなときに図 3.5 の PL 反映ボタンは，プライオリティレベルに応じてタスクの縦軸位置を変更させる。また矢印のボタンはタスク間をその方向に広げたり縮めたりすることが出来る。

このようにして，タスクグラフを通じて処理内容の構造的な特徴を容易に捉えることが出来る。

また元に戻したい場合には，メニューを選択するとタスクの配置のみを戻す機能と，配置・倍率を戻す機

図 3.5　タスクグラフ配置変

能を実行できるようにした(図 3.5 参照)。

③　マクロタスクグラフの描画[4]

タスクの中でも階層的にタスクを含むようなタスクをマクロタスクと呼ぶ。すなわちマクロタスクは１つのプログラムを構成するタスク X の中に X を構成する複数のタスクが存在することを示す。そして上位層と同様に，このタスク X を構成するタスク間には先行制約があり，その先行関係を表したものがマクロタスクグラフである。

マクロタスクは通常タスクとは違い青い色で表示されており，それをダブルクリックすることでそのマクロタスク用のタスクグラフを読み込み，新たなウィンドウで表示する。

3. 3　ガントチャートツールの基本機能[5]

図 3.6 はガントチャートの概観である。スケジューリング結果ファイルを読み込むと，このようなガントチャートが表示される。

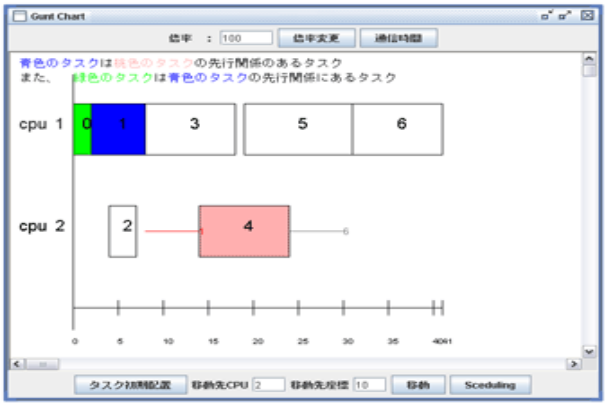

図 3.6　ガントチャート概観

①　拡大，縮小機能

ガントチャート上部にある入力フォームに数字を入力すると，その値の倍率でガントチャートを表示する(図 3.7 参照)。

図 3.7　倍率変更と全体通信時間表示

図 3.6 では全体像がウィンドウ内に収まっているが，CPU 数が多い場合やタスク数が多くて全体が見られない場合にこの縮小機能を使えば全体が一見できる。

②　先行タスクの表示

あるタスクをクリックすると，そのタスクの先行タスク，先行タスクのさらに先行タスクを色を分けて表示する。同時に，そのタスクに関係した通信情報も表示する。受信情報は赤，送信情報は灰色で表示する。

図3.6ではタスク4をクリックしており，タスク4の先行タスク1が青く変化し，タスク１の先行タスクである０が緑色に変化している。また，タスク１から４への通信が赤いラインで，タスク４から６への通信が灰色のラインで描かれており，通信が発生する先行タスクや後続タスクもわかるようになっている(図3.8参照)。

先行タスクが解る事で，もし通信を必要としない位

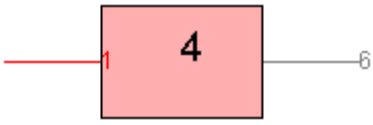

図 3.8　選択タスク

置にタスクを移動することが可能なら，それにより通信の削除などが出来る。また，後続タスクに関しても同じことが言える。指定したタスクから発生している通信を削除できるような位置にタスクを配置できれば，通信による後続タスクの実行開始の遅れを除去できる。

先行制約の流れ上にある全てのタスクはなるべく同じ CPU に割り当てることで，通信を発生させずに実行が出来る。その為先行タスクや通信の情報はタスクを手動で再割り当てをする場合にこの機能が重要になってくる。

③　通信時間表示

通信時間表示チェックボックスにチェックをいれると，全ての通信時間を赤いラインで表示する(図 3.9 参照)。チェックボックスにチェックが入っている間は必ず通信時間が表示され続ける。

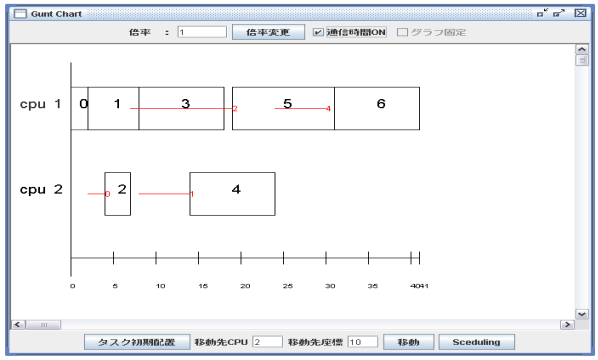

図 3.9　通信時間表示

ここでは，CPU がデータの送受信とタスクの計算は同時に実行ができるが，送信同士または受信同士は同時には行うことが出来ないことを前提としている。したがって，あるタスクが複数の先行タスクを持ち，それらが別の CPU に割り当てられている場合はそれらの通信は逐次的に処理されることになる。スケジュー

リング結果では，全ての先行タスクからの受信完了以降にそのタスクの開始時刻が定められており，それをガントチャートで確認することが出来る。

④　タスク移動による再割り当て機能

タスクをドラッグ＆ドロップして，移動したい場所にタスクを置く事により，指定した CPU および時刻にタスクの割り当てを変更する機能である。先行制約と通信の配置を考慮して割り当て位置の再計算が行われる。

また，ドラッグ＆ドロップではなくクリックしたタスクを次のように指定する方法で再計算する手段も準備した。ガントチャートの下部にある移動先 CPU 欄に移動先 CPU，移動先座標欄に時刻を入力し，移動ボタンを押すと選択済みのタスクをその場所に移動することが出来る（図 3.10 参照）。

図 3.10　タスク移動指定バー

タスクを移動するとそれに伴い後続タスクが，通信や処理の開始時刻が再計算されて正しく配置される。図 3.11 は図 3.6 の状態からタスク 3 を CPU1 から CPU2 に移動させている。

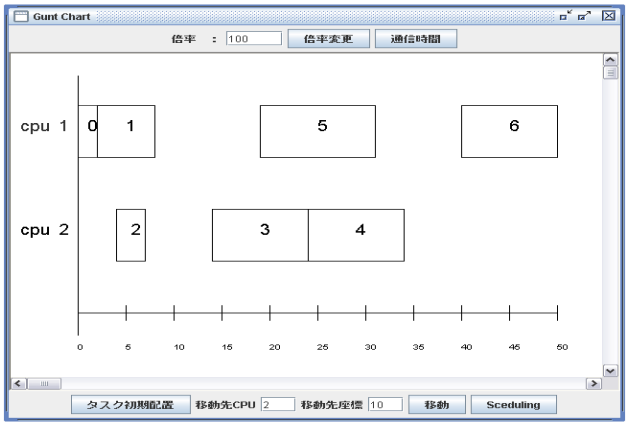

図 3.11　タスク 3 の移動

この機能により，今表示されているスケジューリング結果から特定のタスクを別の CPU や時間に移動した場合の情報を作成して確認，考察することができる。

割り当ての際，もし指定したタスクの処理開始時刻が先行制約の関係上その時間に配置することが出来ない場合がある。このときは移動先の CPU で一番早い配置可能な時間に実行開始時刻を自動設定する。すでに別のタスクが割り当てられている場所に割り当てようとした場合は，タスク情報の再計算の際にその割り当て済みのタスクの終了後に実行を開始するように移動するタスクの実行開始時刻を変更する。移動したタスクにより影響を受ける後続タスクを始めとする全てのタスクの配置も再計算され，描画し直される。

⑤　変更データの保存

再割り当てなどで変更されたスケジューリング結果を新たにファイル名をつけて保存する。

⑥　タスク初期配置

タスクを最初の状態に戻す機能である。この場合の最初とは，ガントチャートウィンドウが表示されたときの最初のスケジューリング内容を示す。

3. 4　ガントチャートとタスクグラフの連携

ガントチャート，タスクグラフをそれぞれで表示するだけであれば，一つにまとめる必要は無い。このツールの最大の特徴の一つに，ガントチャートとタスクグラフの表示を連携させる機能がある。

①　クリティカルパス表示

タスクグラフでクリティカルパスを表示すると，ガントチャートでクリティカルパスのタスクの色を変化させる（図 3.12 参照）。

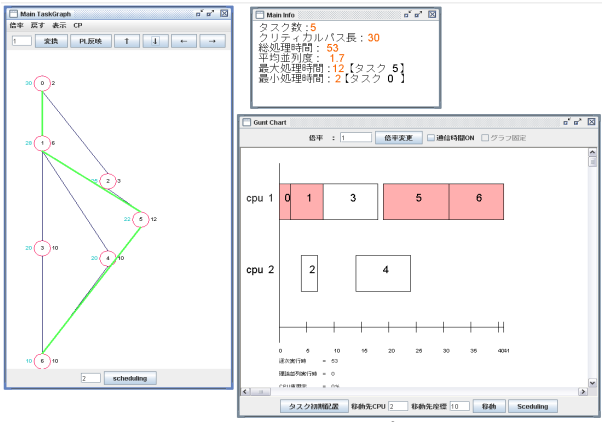

図 3.12　クリティカルパス

クリティカルパスの情報はタスクグラフが持っている。したがって，クリティカルパスの表示をする度にクリティカルパスのタスク情報をガントチャート側に送信する。ガントチャートではその情報を元にクリティカルパスのタスクの色を変化させる。これにより，クリティカルパスのタスクがどの CPU に割り当てられているのかが一見してわかる。クリティカルパス長は並列実行時間の下限値であるから，そのパス上のタスクが同じ CPU に割り当てられていないと，通信の発生により明らかに実行時間がクリティカルパス長より長くなる，といったことが確認できる。

②　CPU・時間軸での表示

ガントチャート上で CPU，時間軸をクリックした場合にはタスクグラフ上でもそのタスクの色が変化する。

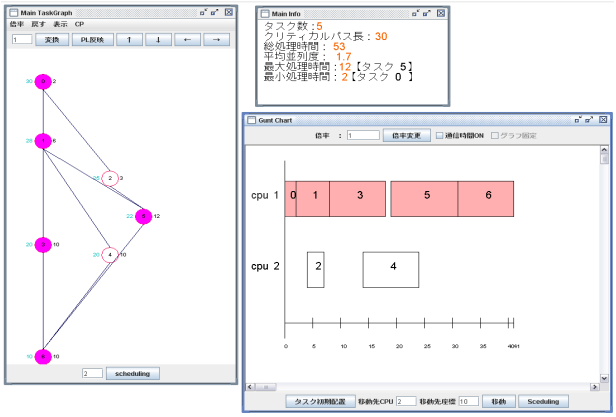

図 3.13　CPU1 選択時

図 3.13 では，ガントチャート上で CPU 1 をクリックしている。すると，ガントチャートでは CPU1 に割り当てられた全てのタスクの色が変化して，同時にタスクグラフにそれらのタスク番号を送信して，タスクグラフ上でも同一 CPU に割り当てられているタスクを一覧できる。この色の変化は CPU をクリックしている間続き，マウスのボタンを離すと元の表示に戻る。

タスクの少ない状態ではこの機能はあまり必要性を感じられないかもしれないが，タスク数や CPU 数が増えるとどのタスクがどこで実行されていて，タスク実行がどの様な流れになっているのかが分かり易くなる。

③　リストスケジューリング

通信を考慮せず，探索を行わないでヒューリスティックのみに基づいたスケジューリングを行う。ツールのリストスケジューリングには二種類あり，全体をスケジューリングするものと選択したタスク以降を再スケジューリングするものがある。

タスクグラフ側にあるスケジューリングボタンを押すと，左側の入力フォームの CPU 数で全体をスケジューリングし，その結果を新しいウィンドウでガントチャートに表示する。（図 3.14 参照）

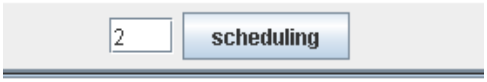

図 3.14 全体スケジューリング（タスクグラフ）

ガントチャート側にあるスケジューリングボタンでは，ガントチャートでクリックして選択したタスクよりも後の時刻に始まるタスク全てを再スケジューリングするようになっている。もし選択したタスクと同時刻に始まるタスクが存在した場合にはそのタスクも再スケジューリングの対象となる。

スケジューリングアルゴリズムには CP/MISF 法（Critical Path/Most Immediate Successors First 法[1]）を使っている。これは通信を考慮していないヒューリスティックアルゴリズムで，リストスケジューリングの中で割り当てタスクを選択する際に，タスクに優先順位を決めて利用するものである。すなわち，レディタスク数がアイドル CPU 数より多いとき，どのレディタスクから優先的に割り当てるかを選ぶために次のようなヒューリスティックを用いる。

1）各タスクから出口ノードまでの最長パス長を計算する。

2）最長パス長の大きい順にプライオリティリストを作る。

3）もし最長パス長が等しければ，その中で直接後続タスクの多い順に，プライオリティリストを作成する。

4）先行タスクの実行が終わり，実行可能（レディ）となったタスクのうちで，プライオリティリストにおける順位が高いものから，空いているプロセッサ（アイドルプロセッサ）に割当てる。このとき，レディタスクが存在するかぎり，プロセッサをアイドル状態のままにしておくようなことはしない。

全体スケジューリングにヒューリスティックを用いて探索解法を行わないのは，単純に処理時間を短く抑えるためである。また部分スケジューリングの方はタスクを手動で移動する作業を行った後，残りのタスクについて簡単なスケジューリングを適応したい場合に役立つ。

3. 5　タスクスケジューリングのリアルタイム表示

タスクスケジューリングの解は，分枝限定法のような全探索により求められるが，膨大な処理時間がかかるため探索の初期段階でできるだけ良い解にたどり着けるようにヒューリスティックを入れることが多い。そのヒューリスティックが実際にどのくらい効いているかを調べるためには，タスクスケジューリングの解を探索する上で，どのような順番で探索がされ，枝切りを行ったかなどの情報はとても重要である。

そのような情報をスケジューリングプログラムから貰い，リアルタイムで表示する機能を加えた。この機能により，スケジューリングプログラムがどの様に暫定解を出しているのか，その流れが判るようになり，探索アルゴリズムを改善する上での補助になる。

3. 5. 1　スケジューリングプログラム側処理

スケジューリングプログラム側は DF/IHS 法[1]を使い，解の探索を行う。DF/IHS 法は CP/MISF 法のヒューリスティック効果を，分枝限定法（Branch & Bound）に取り入れた，ヒューリスティックな探索アルゴリズムである。

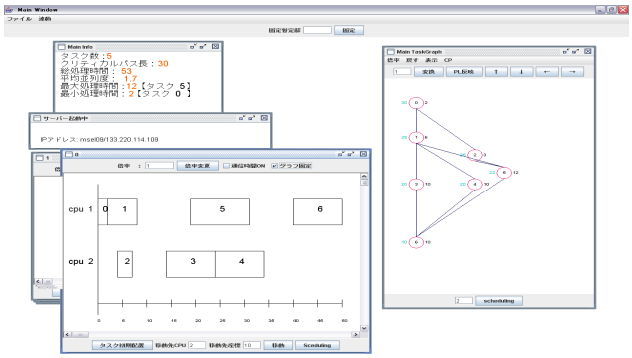

図 3.15　探索プログラムとの連係機能

スケジューリング側は探索中に暫定解が得られる度にその結果を図 3.1 の右側のようなフォーマットで本研究のツールに送信してもらう。

3. 5. 2 ツール側処理

図 3.15 は本ツールがスケジューリングプログラムと通信中の一例である。スケジューリングプログラム側から暫定解のデータを受信するのを待ち，受信したらそれをガントチャートで表示する。表示が終わったらまた受信待ちになる。以上をスケジューリング側から探索終了の合図が送られてくるまで続ける。スケジューリング側から受信したファイルはユーザの指定する場所に自動で名前をつけてテキスト形式で保存する。ツールはユーザが指定した数(1～10 枚)だけウィンドウを複数表示し，暫定解で何処が改善されたのかを確認することが出来る。暫定解は自動で割り振られたウィンドウの番号の順番で書き換えられてゆくが，ウィンドウ上部の『表示固定』チェックボックスにチェックをいれることでそのウィンドウの情報の書き換えをしないようにすることが出来る。また，暫定解の情報の書き換えが速く，チェックをする余裕がない場合はメインウィンドウの上部にある入力フォームにあらかじめ整数値を入力する。すると，その番号の暫定解は固定され，情報が上書きされないようになる。これにより基準のデータを決めて，そこから次の暫定解がどこを改善したのか，という情報を得ることが出来る。

4．評　価

ツールの評価のために，既存のヒューリスティックを二つ取り上げ，それらのスケジューリング結果を比較する。一つはリストスケジューリング，もう一つは CP/MISF 法である。後者は前者を改善したアルゴリズムであり，その改善点を本ツールで明らかに出来ることを示す。ここでは，図 4.1 のタスクグラフを利用し，リストスケジューリング法と CP/MISF 法を使ってスケジューリングを行った。

リストスケジューリングを行った場合，図 4.2 のようなスケジューリング結果となり，CP/MISF 法を使うと図 4.3 のスケジューリング結果となる。

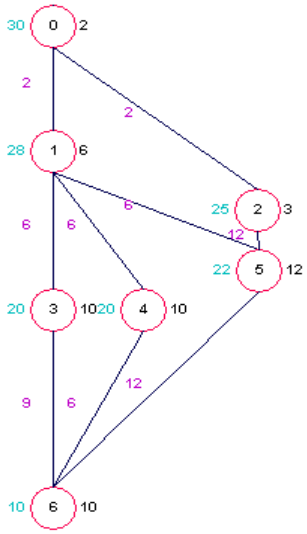

図 4.1　使用タスクグラフ

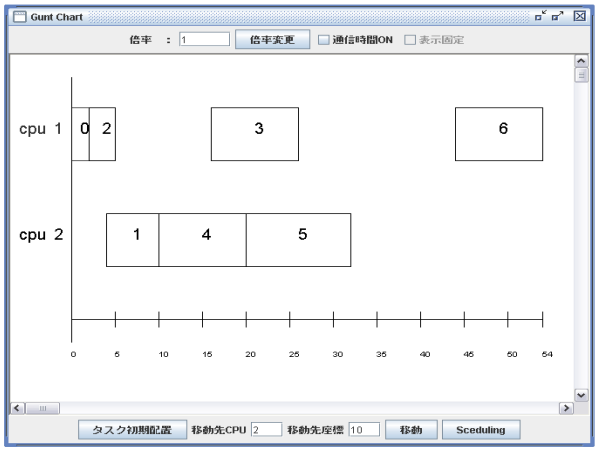

図 4.2　リストスケジューリングによる結果

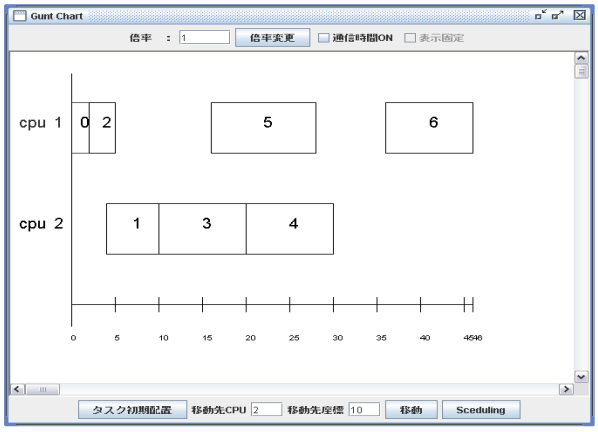

図 4.3　CP/MISF 法による結果

図 4.2 と図 4.3 ではタスク３とタスク５の割り当てが全く異なる。

リストスケジューリングではタスク番順に割り当てているので, タスク５の割り当ては後回しなるが, CP/MISF 法では出口ノードまでの最長パス長の長い順にタスクを割り当てているので，タスク５のプライオリティレベルの高さにより割り当てが早くなっているからである。リストスケジューリングから CP/MISF 法へと近づけるために，まず全体の実行時間が延びている原因を調べる。タスクの通信情報や先行タスク関係を確認すると，タスク５の先行タスクがタスク１とタスク２であることが分かる。つまり，タスク１とタスク５の間にタスク４が入ってしまっているためにタスク５の実行開始が遅れていることが分かる。そこで，タスク４をタスク５の後ろに移動する。また，通信時間を確認すると，タスク５からタスク６への通信時間は他の通信に比べて長い。この通信はなるべく避けたいので，タスク５は CPU1 に移動させる。そして，この場合にはタスク５の処理をタスク３よりも先行させる必要があることが分かり，ガントチャートおよびタスクグラフに表示されている情報から，プライオリティレベルの高いタスクを先に割り当てるべきであることが分かることになる。

このようにして通信時間や先行タスクを確認し，タスクを移動させてゆくことによってより良い解を探し出すことが出来る。

以上より，ツールがスケジューリングの考察の手助けけが出来ることが確認できた。

５．おわりに

従来，独立していたタスクグラフ描画ツールとガントチャートツールを統合し，それぞれのツールの連携をとることができた。このツールの開発により，タスク情報やスケジューリング結果を確認しながら手動でのスケジューリングの考察が簡単になった。また，リストスケジューリング機能とガントチャートのタスク再割り当て機能により，手動による再スケジューリングも可能になった。たとえば，プライオリティレベルの近いタスクが並列に処理されているかどうかをタスクグラフやガントチャートで確認し，もし並列ではないのであればそこを改善すればより良い解が求まる可能性が高くなる。

また，スケジューリングプログラムと通信し，その暫定解の流れをガントチャートで確認することで視覚的に現在のアルゴリズムの解法を確認することが出来る。そうすることで新たにヒューリスティックな考察をしてアルゴリズムを改善する手助けになる。今後はスケジューリングプログラムとの連携を重視した開発が必要となってくるだろう。スケジューリング側からデータをもらってきて表示するだけではなく，途中結果を確認してユーザ側で手動のタスク移動などで変更した順番をスケジューリング側に送信，その情報を元にスケジューリングを進めるなどの連携が考えられる。

参考文献

[1] 笠原博徳 : “並列処理技術”， コロナ社，1991

[2] 牧野勝好，宝達真吾 : “タスクスケジューリング用 GUI 環境の製作”,2005 年度成蹊大学工学部卒業論文

[3] 野本利信 : “タスクスケジューリング補助ツールの開発・拡張―タスクグラフのマルチウィンドウ表示と GUI 機能の拡張―”, 2006 年度成蹊大学工学部卒業論文

[4] 大藪俊通 : “タスクスケジューリング補助ツールの開発・拡張 ―タスクグラフのマルチウィンドウ表示と GUI 機能の拡張―”, 2006 年度成蹊大学工学部卒業論文

[5] 高山翔 : “タスクスケジューリング補助ツールの開発・拡張 ―タスクグラフ情報ファイル出力とガントチャート追加機能の実装－”, 2006 年度成蹊大学工学部卒業論文