メタ戦略の評価分析と
並列 Tabu Search アルゴリズムの提案

小樽商科大学大学院平成12年3月修了
現㈱アジェンダ　丸　田　寛　之
小樽商科大学　加　地　太　一

1　はじめに

対象が組合せ的，離散的な条件の下での最適化を組合せ最適化問題と呼び，配送計画問題，生産スケジュール問題および工場施設配置問題など多くの意思決定の問題で利用されている。しかし，この問題の特徴として問題のサイズにしたがって組合せ的爆発をおこし，計算機でも数兆年の時間を要する計算困難な問題となる。そこで，近似解を求め代替する戦略として，近年メタ戦略の有効性が示されている。これまで様々なメタ戦略の改良アルゴリズムが提案されてきた。しかしながら，実際的な利用に際してはそれらの改良アルゴリズムよりも基本的な構成のアルゴリズムを用いることが一般的であるが，これらの基本的な構成のアルゴリズムの優劣は明確ではなく，各応用分野における利用に際して総合的な評価を行い特質を明らかにすることが求められる。ここでは組合せ最適化問題の代表的な問題であるグラフ分割問題を対象として，基本的な構成による各種メタ戦略を複雑度の高い複数分割問題に適用し，評価実験を行い各アルゴリズムの基本性能を比較評価するものである。

また，これらのメタ戦略は計算機の進化とともに飛躍的な発展を遂げてきた。しかしながら最近は従来の計算機アーキテクチャではすでに計算能力が頭打ちの状況であり，新しいパラダイムとして並列計算機が注目されている。これは従来の逐次的なアルゴリズムとは異なるアプローチが要求されるものである

が，並列計算機は複数のプロセッサで処理を複数同時に行うことで計算性能を大きく向上させるものであり，そのメリットは非常に大きい。そこで，探索時間を要する Tabu Search を並列化し探索時間及び解の質の改善を試みた。Tabu Search は探索する近傍空間のサイズが大きくなると多大な計算時間を要する。そこで近傍探索を分割することで複数のプロセッサで分散して探索することによる計算時間の短縮を試みた。さらに，各プロセッサ上で部分解を同時に複数生成し，Tabu Search による局所的な探索を複数同時に行うことで解の質の向上を試みた。これは問題を独立した部分問題に分割し，各プロセッサ上で部分問題に対して探索を行い収束させ，部分解を生成する。各プロセッサで生成された部分解は通信によって完全解へ合成する。これを繰り返すことで収束性に優れ，探索時間も短いアルゴリズムを提案した。このアルゴリズムを構成する上で，Token Passing という手法を用いた。この手法では計算環境にとらわれずに比較的自由な実装を可能とし，効果的に各プロセッサにスケジューリングを行うことができる。各プロセッサには完全に独立した部分問題が割り当てられ，独立した探索を行うことが可能なことから，各プロセッサは他のプロセッサと異なるアプローチで解を収束させ，同時に他のプロセッサの状態に依存せずに非同期に探索を行うことも実現できるものである。

2 グラフ分割問題

グラフ分割問題は，NP-困難な組合せ最適化問題の一種であり，VLSI の回路設計等の CAD や配置・配送問題，プログラム分割問題などに応用されるものである。この問題はウェイトをもつ n ノード（頂点）とコストをもつエッジ（辺）からなるグラフを各部分集合に分割し，その異なる部分集合間のエッジコスト総和が最小にする問題である。ノード数 n が十分に小さい場合，厳密解法によって解くことができるが，ある程度を超えると組合せ的爆発により厳密解を得るのは難しくなる。また本問題は多分割のため，従来研究されている 2 分割に比べて汎用的であるが複雑度が高くなる。

今回対象とする問題は，頂点集合 V と無向辺集合 E からなるグラフ $D(V, E)$ において，そのカットエッジのコストが最小となるような多分割を求める問題であり，制約条件として分割した各部分集合における頂点のウェイト総和がある値 P_i をこえないものとする。この問題は以下のように定式化される。

$$\min \sum_{i \in V_i, j \in V_j, i \neq j} e_{ij}$$

$$\text{subject to} \sum_{v \in V_i} w(v) \leq P_i,\ i = 1, \ldots, k$$

ここで，$w(v)$ は頂点 v のウェイトとする。また，ブロックサイズ P_i と分割数 k は与えられた数である。

3　メタ戦略

組合せ最適化問題は NP-困難な問題とされ，厳密な完全解を求めることは計算量の問題から非常に困難である。そこで何らかの近似解法によって問題を解き，近似解を求めることになる。

近年では組合せ最適化問題を解くためにヒューリスティックな知識を有機的に組合わせて，より高度なアルゴリズムを構成したメタ戦略の研究が盛んに行われており，従来の近似解法を超えたパラダイムとして注目されている。メタ戦略はヒューリスティックにパラメータを追加し，その自由度を用いて問題を解くテクニックであるため，パラメータを適正値に設定する作業が不可欠となる [1]。これらのメタ戦略はこれまで改善アルゴリズムも多く提示されている。しかし，問題解決のための実際的な利用では，通常は標準的な構成で利用されるのが常である。しかしながら，各種メタ戦略の標準的な構成の能力はあまり知られていないため，実際的な評価を行いその特質を明らかにする。ここではまずメタ戦略の基本的な考え方とグラフ分割問題への適用における構成を示す。

3．1　近傍の構成

メタ戦略においては，そのほとんどが近傍を基礎として構築されており，近傍構成の設計が重要となる。近傍の構成は問題依存であり，各問題に適用するには問題の特性に合わせて設計する必要がある。

近傍 N は実行可能解の集合 X を与えたとき，以下の写像として定義される。

$$N : X \rightarrow 2^X$$

すなわち，実行可能解の集合から，そのべき集合（部分集合の集合）への写像が近傍である。実行可能解 $x \in X$ で，$f(x) \leq f(y)$，$\forall y \in N(x)$ を満たすものを，近傍 N に対する局所的最適解（local optimal solution）と呼ぶ。[1]

ここで用いるグラフ分割問題の近傍は通常用いられるブロック間の頂点の交換に基くものを使用する。このとき，解 x を

$$x = (S_1, S_2, \ldots, S_n)$$

とし，S_i は分割された頂点の部分集合を表すものとして，この解 x に対して近傍構造 $N(x)$ を以下のように定義する。

$$N(x) = \{x' = (S_1, S_2, \ldots, S'_v, \ldots, S'_w, \ldots, S_n) : S'_v = (S_v \setminus \{i\}) \cup \{j\},$$
$$S'_w = (S_w \setminus \{j\}) \cup \{i\}, \text{ for } i \neq j, \ \forall i, \ \forall j\}$$

3．2　Local Search

Local Search はメタ戦略の基本となるアルゴリズムであり，今回対象とするアルゴリズムも Genetic Algorithm を除いてはこの Local Search を基本的な構造としている。今回実験を行った Local Search は最も基本的なものである。探索はまず現在の解の近傍の中から，最も解を改善するものを選択する。これを次の解として採用し，近傍の中にこれ以上改善する解がなくなった時点で探索を終了する。この Local Search の短所は局所最適解に陥りやすいことであり，一旦局所最適解に陥ると，脱出手段を持たない Local Search ではそこで探索を終了してしまう。このため，他のアルゴリズムと比較して解の質は

それほど良くない。Local Search の近傍生成は標準的なものとして，解 x の近傍 $N(x)$ の中から最良な解を選択し，次の解に移動するものとする。この解生成を関数 $imp\ (x)$ と表現し，以下のように定義する。

$$imp(x) = \begin{cases} x'; \text{ if } \min\{cost(x')\} < cost(x), \text{ for } \forall x' \in N(x) \\ \emptyset \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

3．3　Tabu Search

Tabu Search は Local Search の変形で，一度交換した頂点を Tabu List TL に一定期間記憶し，同じ解の探索を禁止することで解のサイクリングを防ぐものである。近傍生成は標準的なものとして，解 x の近傍 $N(x)$ の中から TL に含まれるものを除いたなかで最良な解を選択し，次の解に移動するものとする。これを関数 $impTL(x,\ TL)$ と表現し，以下のように定義される。

$$impTL(x,\ TL) = x' \text{ if } f(x') \leq f(y) \text{ for all } y \in (N(x) \setminus TL)$$

通常，Tabu List は解そのものを保持するのではなく，解の移動に関連する何らかの属性を Tabu の要素とする。今回は移動した頂点を属性として Tabu List に記憶し，一度移動した頂点は一定期間の移動を禁止した。

今回はこれに長期メモリ (Long-term memory) を付加したものを採用した。一般的な Tabu List は Tabu Length として与えられた期間のサイクリングを防ぐものであるが，長期メモリは長期的なサイクリングを防ぐものであり，探索が終了するまで保持される。長期メモリは配列 $LT\ (n)$ に頂点の移動回数を記憶するものとした。このとき，頂点 i と頂点 j を交換したときのコストを以下の式で評価する。

$$f(S_{ij}) + \frac{(LT(i) \times BIAS + LT(j) \times BIAS)}{2}$$

これは，頂点 i と j を交換したときのコスト $f(S_{ij})$ に頂点 i と j の移動回数に応じたコストを評価時に加えることで，頻繁に移動を繰り返す頂点を移動しにくくするものである。

3．4 Simulated Annealing

Simulated Annealingは，局所探索をランダムに行い，解が改悪された場合でも新しい解にある確率で移ることで，局所最適解に補足されることを防ぐアルゴリズムである。

このアルゴリズムは，「温度」をゆっくりと低下させることで結晶構造を安定させる焼き鈍しのアナロジーを用いている。Simulated Annealingでは，近傍から新しい解を確率的に選択する。選択された解は，現在の解よりも改善されるものであれば無条件に受理（Accept）される。改悪するものであった場合，次式のように温度を用いて確率的に受理するかどうかを決定する。

$$\begin{cases} 1 & \text{if } f(x') \leq f(x) \\ \exp\left(\dfrac{f(x)-f(x')}{c}\right) & \text{if } f(x') > f(x) \end{cases}$$

cは温度であり，$f(x)$は現在の解のコスト，$f(x')$は新しい解のコストである。これはつまり，解が改悪される場合は$e^{-\Delta/c}$の確率で受理することを意味する。この温度cは一定の係数によって減少していく。温度が高いときは改悪される解は受理されやすいが，温度が低くなるにつれ改悪される解は受理されにくくなる。

3．5 Genetic Algorithm

Genetic Algorithmは自然淘汰や突然変異といった，生物の進化論をモデルとしたアルゴリズムである。Genetic Algorithmでは，Reproduction，Crossover，Mutationの3つのオペレータによって遺伝子を進化させるものである。このアルゴリズムは，集団として多数の解を個体として保持する。Reproductionでは，これらの集団から目的関数値の高い個体を高い確率で複製する。次に，Crossoverによって個体同士を交叉し，それぞれのより良い部分を合成して新しい個体を生成する。また，Mutationでは複製の際にある確率で突然変異を発生させる。これによって個体群に多様性をもたせるものである。

Genetic Algorithm における遺伝子の要素は記号列であり，対象とする問題を記号列として表現しなければならない。今回 Genetic Algorithm で採用した近傍構造は，すでに実績があり一定の結果を示しているものを採用した[9]。各ブロック毎に頂点を線形に配置し，各頂点を遺伝子として扱うものとした。ここで用いる交叉は，親遺伝子から優位な部分集合を確率的に選択し，対応する遺伝子部分を保持する。それ以外の遺伝子部分を他の親遺伝子からコピーするものとした。

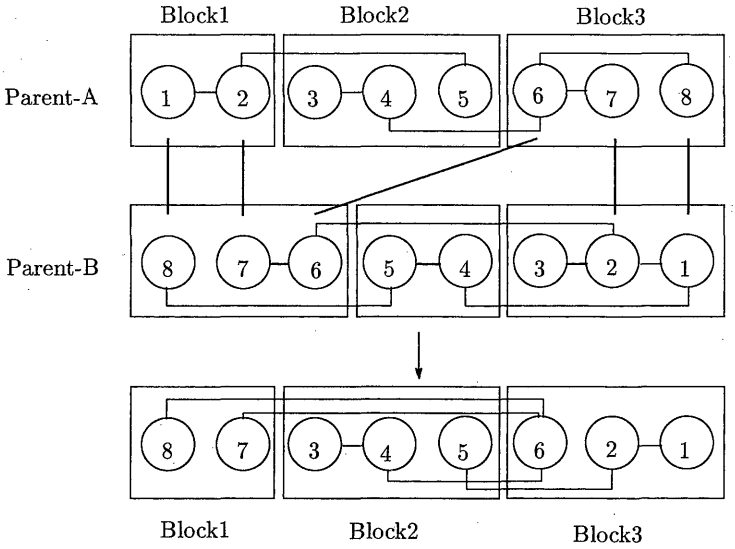

図1 ： Genetic Algorithm Crossover

図1は交叉の一例である。まず，8ノード3ブロックからなるグラフにおいて，各頂点を線形に配置する。Parent-B から Parent-A へ交叉し次世代の解を生成する場合，優位なブロックを Block2とすると，Parent-A の Block2をまず保存し，それ以外のノードを Block-B のものと置き換える。つまり，Parent-B における Block1のノード8と7を Parent-A の Block1へコピーする。これで Parent-A の Block1は埋まったので次は Block2であるが，Block2

は優位なブロックであるため，そのまま保存する。次にParent-Aの優位なブロックに含まれるノードを除いて，Parent-AのBlock3へParent-Bからノードを同様にコピーする。結果，Parent-Aの優位なブロックを保存したまま，Parent-Bの情報と交叉できることになる。

また今回採用した親遺伝子の選択は，個体のコスト値に応じて優秀なものから確率的に選択する，適応度比例方式（ルーレット選択）とした。このとき使用した適応度関数は次式である。

$$Fitness = \frac{1}{cost}$$

つまり，コストが小さいほど適応度が高いものとした。また交叉点の選択は，選択した親が保持する解のなかで最も優秀なブロックを保存するものとした。このとき，各ブロックの内部コストが大きいものは外部コスト（カットエッジコスト）が小さい傾向であることから，内部コストがもっとも大きいブロックを保存するものとした。また，突然変異は指定した確率でランダムに選択された頂点をブロック間で強制的に交換するものとし，これを全ての親遺伝子に対して行った。

4　メタ戦略の数値実験による評価分析

今回の各メタ戦略の基本的な構成による実験では，100～500ノードのランダムグラフを対象とした。ランダムグラフの生成は，ノードxから$x+1$へのエッジを全ノードに対して生成することで，closedなグラフであることを保証し，さらに各ノードからノード数の10%づつランダムにエッジを発生させた。よって，nノードグラフのエッジの総数は$n \times \{(n/10)+1\}$である。また，各エッジのウェイトは1～10の範囲でランダムに設定し，各グラフにおける各ノードのウェイトは固定とした。

メタ戦略の重要な要素としてパラメータの設定があり，これによって解の質が大きく左右される。通常，これらの値は事前の数値実験によって設定を行う

ものである。Tabu Search では，Tabu Length の設定が必要である。Tabu Length は事前に500ノードのグラフでいくつかのパラメータで実験を行った。この予備実験は異なる10種類のグラフ，グラフ１～グラフ10に対して行い，その平均から20が最良であったため Tabu Length として採用した。また，Longterm memory の BIAS 値も同様に実験を行ったところ，0.20が最良であったため，0.20に設定した。Simulated Annealing に関しては初期温度，温度減少係数，終了基準温度，内側ループの回数，その増加係数の設定が必要である。初期温度は高いほど良いコストが得られるが，高すぎると無駄な探索が増え，計算時間が増加する。初期温度と終了基準は Acceptance Ratio と同時に計測し，Acceptance Ratio が0.7～0.0003となるように初期温度を15，終了基準を0.01に設定した。つぎに温度の減少係数は１に近いほどゆっくりと冷却されることになるため，１に近いほどコストが改善されるが，これも同様に大きすぎると計算時間が劇的に増加する。これも同様にいくつかの係数に対してコストと計算時間を計測したところ，0.97まではコストの改善が見られるが，それ以上はコストは改善されず計算時間だけが増加するため，0.97を採用した。また，内部ループの回数は100とし，増加係数は計算時間とのトレードオフで1.03とした。Genetic Algorithm では Population 数と突然変異発生率を設定する必要がある。Population 数は個体群に含まれる個体の総数であり，多い程良いコストが得られる傾向にあるが，多い程計算時間が増加する。これもいくつかの Population で実験を行った。Population を増加させるほどコストが低下するが，計算時間も劇的に増加する。実験の結果，300までは減少傾向を示したが，それ以上は停滞傾向であったため300を採用した。また，突然変異発生率も同様に実験を行い，最良であった0.04を採用した。これは１イテレーションにおける各個体に対する確率である。

今回実験に用いた環境は Intel Celeron Processor/400MHz を搭載した PC/AT 互換機の Linux 上で実験を行った。表１に８ブロックの分割問題における実験結果を示した。Simulated Annealing，Genetic Algorithm に関してはアルゴリズム中に確率要素があるため，それぞれ10回実験を行い，その平均

表1 : Computational Result obtained using LS, TS, SA and GA

Algorithm		100	200	300	400	500
Local Search		3838.0	16526.0	38252.0	69282.0	108484.0
Tabu Search		3812.0	16335.0	37710.0	68843.0	108055.0
Simulated Annealing	AVG	3704.8	16269.6	37794.0	68968.8	108425.6
	MIN	3687.0	16238.0	37744.0	68770.0	108302.0
	MAX	3734.0	16342.0	37879.0	69104.0	108526.0
Genetic Algorithm	AVG	4491.4	18559.4	42671.0	75423.0	118619.4
	MIN	4346.0	18438.0	42580.0	75204.0	118457.0
	MAX	4558.0	18647.0	42765.0	75822.0	118936.0

表2 : Computational Time obtained using LS, TS, SA and GA (sec)

Algorithm	100	200	300	400	500
Local Search	1.40	24.58	137.78	756.56	1391.83
Tabu Search	7.25	93.66	423.36	1357.88	2210.26
Simulated Annealing	29.49	75.41	147.51	230.50	307.12
Genetic Algorithm	110.81	500.45	974.51	1814.66	1886.03

をAVGに，最小値をMINに，最大値をMAXに示した。また，表2はその計算時間を示した。

Local Searchは，Tabu SearchやSimulated Annealing等と比較してそれほど良いコストは得られなかった。このアルゴリズムは局所最適解に陥ると脱出手段をもたないため，これ以上の改善を行うことができなくなる。最初に局所最適解に陥った時点で探索を終了してしまうことから，他の局所最適解からの脱出手段をもつアルゴリズムと比較して解の質が劣ることは明らかである。

Tabu Searchは比較的良い結果を出しているが，今回採用したTabu Searchの近傍は全近傍を探索する必要があるため，計算時間が大きく，ノード数の増加とともに計算時間が増加する。このため，Tabu Searchは比較的少ないノード数のグラフにおいては効果的であるが，ノード数が多いグラフに対しては現実的ではない。

Simulated Annealingに関してはTabu Searchと同レベルのコストが得られた。少ないノード数のグラフにおいては計算時間が他のアルゴリズムよりも

長いが，比較的大規模なグラフに対しては他のアルゴリズムよりも圧倒的に短い計算時間で探索を完了する。

Simulated Annealing の近傍生成は，確率的にノードを選択するものであるため，他の Local Search や Tabu Search のように全近傍を探索するものに比較すると圧倒的に短い時間で1イテレーションを完了する。さらに，ノード数やブロック数の影響も受けにくい。実験結果で示される，ノード数に対する計算時間の増加のほとんどは Simulated Annealing のアルゴリズムそのものではなく，ノードの交換処理などのグラフ操作に関する部分が大部分を占めるのが原因である。

また，Genetic Algorithm は今回の標準的な構成のアルゴリズムにおいては Local Search にも及ばなかった。Genetic Algorithm に関しては近傍構造の改善等で若干改善が得られる可能性もあるが，本質的に本問題のような組合せ最適化問題には適用しにくいアルゴリズムであり，Local Search などの局所性のあるアルゴリズムとのハイブリッドが有効であろう。

5　並列処理と MPI

これまでのアルゴリズムは逐次的に実行を行う，いわゆるフォンノイマンマシンと呼ばれる逐次計算モデルに基いたコンピュータで計算されるものである。このモデルはシンプルである反面，単一のプロセッサで全ての処理を行うため処理能力はそのプロセッサの能力に依存する。これまで組合せ最適化問題を解くためのアルゴリズムはコンピュータの進化とともに発展してきた。しかしながら単一のプロセッサの性能はすでに限界に達しており，ベクトル計算機等のスーパーコンピュータもすでに頭打ちの状態で，並列化によって性能を確保しているのが現状である[2]。並列処理は複数のプロセッサを協調させて利用することで計算時間を短縮させる新しいパラダイムであり，組合せ最適化問題に対する新たな近似解法を展開するものであろう。

今回の実験では，並列処理を実現する手段として MPI（Message Passing

Interface）ライブラリを採用した。MPIはメッセージ通信を行うためのライブラリ標準であるが，その利点はポータビリティと使いやすさにある。MPIは分散メモリのマルチプロセッサ，ワークステーションのネットワーク，またこれらの組み合わせの環境でも動作するように設計され，さらに共有メモリでも実現可能である[3]。

今回並列アルゴリズムの実験を行った環境は，いくつかの独立したコンピュータをネットワークに接続し，それをプロセッサ間の通信路として利用するもので，ワークステーションクラスタの一形態である。そのネットワーク構成は，100BASE-TX Ethernetを用い，各コンピュータをFastEthernet Switching HUBへ接続した。コンピュータはIntel CeleronProcessor/400MHzを搭載した同性能のコンピュータを4台使用し，Operating SystemにはLinuxを採用した。

この環境においてMPIの通信パフォーマンステストを行ったところ，1024バイトブロックの実効転送速度は1.3Mbps程度であった。また，ブロックサイズを大きくしたピーク性能でも23Mbps程度であった。この通信路のバンド幅の狭さは弱結合であるワークステーションクラスタの欠点であり，専用の並列計算機等と比べて大きく劣る点となっていることから，アルゴリズムを構成する際に通信量をできるだけ抑えることが求められる。

6 近傍分割並列 Tabu Search

Tabu Searchは全近傍を探索する性質上，多大な計算時間を必要とし一定のノードを超えると現実的な計算時間では解を得ることができなくなる。そこで，近傍分割並列Tabu SearchはTabu Searchの近傍探索プロセスを並列化することで探索時間を短縮するものである。

このアルゴリズムでは，プロセス数p個において探索する近傍空間$N(x)$を$N^1(x)$，$N^2(x)$,...,$N^p(x)$に分割し各プロセスに割り当てることで1プロセスあたりの探索する領域を制限することで，探索時間を短縮する。これは探索する

ブロックのペアを各プロセスに割り当てることによって行った。

このアルゴリズムは通常の Tabu Search と同様に全組合せに対して探索を行うが，プロセスあたりの探索範囲が限定されるため探索量を大幅に削減することができる。とくに Tabu Search では交換対象ノードの探索が計算時間のもっとも大きなウェイトを占めており，この部分の改善は大幅な計算時間短縮を期待することができる。

7　Token Passing 並列 Tabu Search

近傍分割並列 Tabu Search では Tabu Search における近傍探索を並列化し，計算時間の短縮を試みたものであるが，確かに計算時間は効率的に縮約できるが解の質は従来の逐次的な Tabu Search と同一である。ここで提案するアルゴリズムは Tabu Search を基礎として高度に並列化することで，探索時間の短縮，及びさらなる解の質の改善を狙うものである。

このアルゴリズムは図 2 に示されるように割り当てを管理する Token を用いて問題を複数の部分問題に分割し，それを各プロセッサに割り当て局所的に Tabu Search による探索を行うものである。さらに SuperMove によって集合的な移動を行い，生成された部分解を各プロセッサ間で互いに交換することで完全解を生成し整合を得る。この種の問題を並列化する際に問題となるのは，分割方法と同期である。近傍分割並列 Tabu Search では近傍を分割して探索を分散させることで計算時間は短縮するが解の質は逐次アルゴリズムと同等である。並列環境において解の質を改善するためには複数の探索を同時に行うことで探索数を増やすことが求められる。しかし，複数の探索を行うと複数の解が生成され，それを同期させ整合性を保たなければならない。

以上の問題を解決するために，このアルゴリズムでは，問題そのものを分割して各プロセスに割り当てるものとし，探索するブロックのペアを割り当てるが，このときにプロセス間での独立性を高め不整合を防ぐために同じブロックが複数のプロセスに割り当てられる重複を防ぐように割り当てる必要がある。

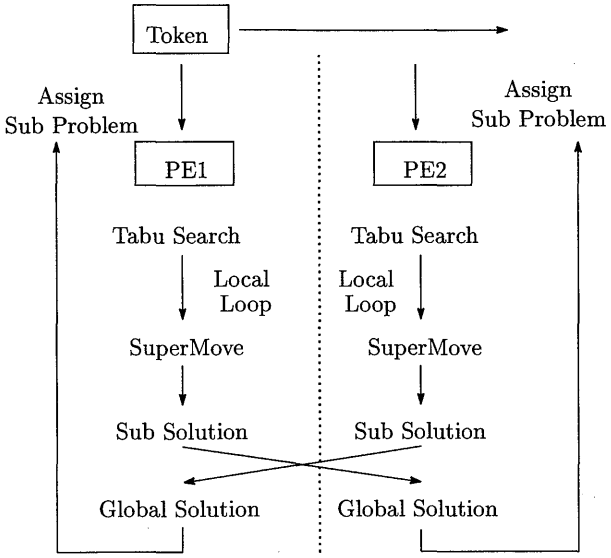

図2 : TokenPassing Parallel TabuSearch

これを実現するために, Token Passing を採用した。Token Passing は Token と呼ばれる配列を各プロセス間で巡回させるものである。Token はブロックの割り当て情報を記録し, Token を持っているプロセスだけがブロック割り当てを行う権利を有する。Token を持つプロセスは Token に記録されている割り当て情報を参照し,割り当てられていない空きブロックを予約する。このとき, Token に割り当て済みのマークをしてから次のプロセスへ送信する。この仕組みにより, 各プロセス間のブロック割り当ての調停時に重複を防ぐものとした。

各プロセス間の解の整合性に関しては, 一定期間の探索を終えたら結果を各プロセスに送信することで整合性を得るものとした。各プロセスは Token から得たブロックのペアを一定期間探索し,その間移動した頂点を記憶しておく。探索終了時に移動した頂点の情報を各プロセスに送信し, 他のプロセスは任意の時点で受信した移動を強制的に行う。このとき, 各プロセスに割り当てられ

ているブロックは独立したものであり，重複は有り得ないため，他のプロセスの探索には全く影響しない。このように各プロセスが非同期で探索を行うことで，同期タイミングにとらわれない自由度の高い探索を行うことが可能となる。

以上のように各プロセスは，Token の取得，ブロックの割り当て，探索，解の同期，Token の開放を繰り返すことで解を収束させていく。探索を他のプロセスより先に終えた場合においても，Token さえ取得できれば新しい探索を他のプロセスと非同期に行えることがこのアルゴリズムの特徴である。この方法では，グラフのブロック数や並列環境のプロセス数，また各プロセスの計算速度などに影響されずに，かつリソースをできるだけ有効に利用することが可能となる。また，この方法によってプロセス間でブロック割り当ての調停を行った場合，各プロセスはそれぞれ完全に独立した部分問題を持つため，各プロセス上で異なる戦略を取ることもできる。また，ブロック割り当て時に各プロセスが持つ戦略に最も適したブロックペアを各プロセスのポリシーにより選択することも可能となる。

今回採用した戦略は，いくつかの実験を行った結果ランダムにブロックを割り当て多様性を持たせ，Tabu Search によって短時間で収束させることが最も良い結果であったためこれを採用した。この戦略の特徴は，Token が一周するだけの割り当てしか行わないことである。例えば，8 ブロックのグラフであった場合，そのブロックのペアの組み合わせは28通りあるが，これを 4 プロセスで探索した場合，4 ペアの組み合わせしか探索を行わないことを示す。探索範囲の限定は解の質を低下させる原因ともなりうるが，このアルゴリズムでは各プロセッサで局所的に探索を強化し，部分解の生成を行うことで解の質を向上させた。

さらに，各プロセスにおいて割り当てられたブロックの探索を終え，同期を取るまえに Super Move を行った。Super Move はブロック内のあるノード集合をまとめて交換するものである。図 3 はその一例である。まず 2 つのブロック内からノードをいくつか選択する。それらの各ノードに接続されているエッジのなかで最もエッジコストが大きいものを 1 つ選択し，その先にあるノー

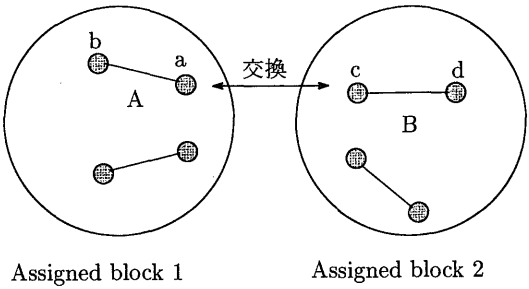

図3：Supre Move

ドを選択する。図の例だとノードaに接続されているエッジで最もエッジコストが大きいエッジAを選択し，その先にあるノードbを選択する。この2つのノードと1つのエッジを1セットとし，ブロックから選択されたいくつかのノードに対してこのセットを生成する。これを2つのブロックに対して行った後，生成されたセットの交換で発生するコスト差を計算する。その中で最もコストを改善するセットのペアを交換する。

この戦略は，例えばノードaのみを移動した場合，エッジAが新たに外部コストとなるが，このコストが大きい場合，ノードa及びノードb単体では移動しにくいことになる。そこで，ノードaとb及びエッジAをまとめて他のブロックへ移動させることによりエッジAを外部コストにすることなくノードを移動させることが可能となる。また，これらのノードを効果的に選択するために，ブロックからノードを選択するときに最も移動回数の少ないノードを選択するというTabu効果とは逆の移動を促進するための選択を行う戦略を採用した。これはLongterm Memoryを用いることで移動回数の少ない頂点を選択することによって実現した。

図4にそのアルゴリズムを示す。x_t^p:= CatchToken(x_t); は，Tokenを取得し解x_tから部分問題を割り当てる関数である。また，SendSubSolution (x_t^p, q; $q=1, 2,...n, p\neq q$); は，プロセッサp上で生成した部分解x_t^pを他のプロセッサqへ送信する関数であり，ReceiveSubSolution(x_t^q, $q=1, 2,...,n, p\neq q$); は，

$t := 0;$

$x_0 :=$ InitialSolution;

repeat

$x_t^p :=$ CatchToken(x_t);

repeat

$x_{t+1}^p := impTL(x_t^p, TL);$

$TL := TL \cup \{x_t^p\} \setminus \{x_{t-TL}\}$

$t := t + 1;$

until local-loop

SuperMove(x_t^p);

SendSubSolution$(x_t^p, q; q = 1, 2, ...n, p \neq q)$;

$x_t^p =$ ReceiveSubSolution$(x_t^q; q = 1, 2, ...n, p \neq q)$;

$x_t :=$ SynchronizeSolution$(x_t^p; p = 1, 2, ...n)$;

until stop-criterion

図 4 ： Token Passing Parallel Tabu Search

プロセッサ q 上で生成された部分解 x_t^q を受信する関数を意味する。$x_t :=$ SynchronizeSolution $(x_t^p; p=1, 2, ...n)$; は各プロセッサ p で生成された部分解を整合し完全解 x_t を生成するものである。

8 並列実験結果

実験に用いたグラフは逐次アルゴリズムに用いたものと同様なグラフ構造である。また，この実験は近傍分割並列 Tabu Search では 2 から 4 プロセッサの，Token Passing 並列 Tabu Search では 4 プロセッサによる比較を行った。

並列 Tabu Search においても，逐次アルゴリズムと同様にパラメータを適切に設定しなければならない。これも同様に事前の数値実験によって設定を行う。

近傍分割型 Tabu Search は Tabu Search の近傍探索部分のみをそのまま並列化したものであるため，必要なパラメータは通常の Tabu Search と同様に Tabu Length の設定が必要である。この値は対象とする問題に対して適切に設定しなければならない。今回は Sequencial と同様に予備実験により20を採用した。

Token Passing による並列 Tabu Search においては，通常の Tabu Length などのパラメータに加えて，ローカルループの回数を設定する必要がある。各プロセッサはまずローカルループで指定されたイテレーションの間，割り当てられたブロックに関して，ローカルで Tabu Search による探索を行う。探索を終えたのちに通信を行い部分解を合成する。このローカルループ数は短すぎると部分解の合成のための通信が頻繁に発生し，粒度が細かくなり全体のパフォーマンスが低下する。しかしながら，長すぎると解の質が低下する傾向にある。500ノード16ブロックと1000ノード8ブロックで実験を行った結果，5が最良であったため5回を採用した。また Tabu Length は通常の Tabu と異なり，部分問題に対する長さであるため，これも同様に計測を行った。この結果，2が最良であったため2を採用した。

8．1 近傍分割型並列 Tabu Search

実験結果を図5に示した。SeqTS は通常の Tabu Search であり，PTS-2p は2プロセッサの，PTS-4p は4プロセッサの近傍分割型並列 Tabu Search である。このアルゴリズムは Tabu Search の近傍探索部分を並列化したものであり，解の質は Tabu Search と同一であるため，探索時間のみを示した。

このアルゴリズムでは近傍探索範囲を各プロセッサに分散させることで計算時間の短縮を狙うものであるが，この近傍探索範囲の分割はブロックのペア単位で行われる。2つのブロックの組み合わせ数が実行環境のプロセッサ数で割り切れる値であれば，計算時間の理論値は1/*processor* である。今回は4プロセッサで実験を行ったが，計算時間は2プロセッサで平均1/1.63に，4プロセッサで平均1/2.98に改善されている。これは並列処理特有のプロセッサ間通信

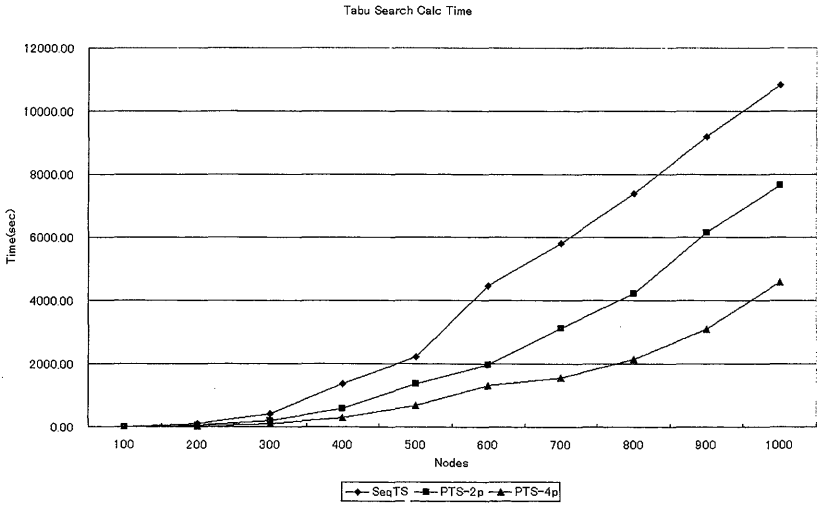

図 5 ：近傍分割並列 Tabu Search Time (sec)

等の処理が入るため，線形に減少はしないものの比較的に効率的に並列化されていることが確認できる。Tabu Search の近傍探索は非常に計算時間を要する処理であり，交換するデータも非常に少ないことから，粒度が粗く通信量の少ないアルゴリズムであるといえる。

8．2 Token Passing 並列 Tabu Search

実験結果を表 3 には100から500ノードのコストを，表 4 には600から1000ノードのコストを，また表 5 には100から500ノードの計算時間を，表 6 には600から1000ノードの計算時間を示した。TS は通常の Tabu Search であり，SA は Simulated Annealing，TP.TS は Token Passing 並列 Tabu Search である。

Token Passing 並列 Tabu Search においては，Sequencial Tabu Search と比較して，計算時間，解の質の両方において改善が認められた。このアルゴリズムでは，近傍探索においてブロックの組合せをプロセッサ数に限定する。従来の Tabu Search の全探索では 8 分割に対して28通りのブロックの組合せに

表3：Computational Result obtained using TS, SA and TP.TS (100-500node)

Algorithm		100	200	300	400	500
TS		3812.0	16335.0	37710.0	68843.0	108055.0
SA	AVG	3704.8	16269.6	37794.0	68968.8	108425.6
SA	MIN	3687.0	16238.0	37744.0	68770.0	108302.0
SA	MAX	3734.0	16342.0	37879.0	69104.0	108526.0
TP.TS	AVG	3787.2	16270.0	37645.0	68653.6	107607.6
TP.TS	MIN	3774.0	16219.0	37583.0	68520.0	107427.0
TP.TS	MAX	3801.0	16334.0	37706.0	68761.0	107761.0

表4：Computational Result obtained using TS, SA and TP.TS (600-1000node)

Algorithm		600	700	800	900	1000
TS		157788.0	216037.0	284427.0	362989.0	448997.0
SA	AVG	157643.2	216348.2	284325.4	362909.4	447511.0
SA	MIN	157549.0	216155.0	284153.0	362369.0	447367.0
SA	MAX	157880.0	216730.0	284828.0	363012.0	447840.0
TP.TS	AVG	156887.2	215295.8	283407.8	361363.8	446172.0
TP.TS	MIN	156746.0	215183.0	283134.0	361204.0	445922.0
TP.TS	MAX	156988.0	215356.0	283778.0	361571.0	446351.0

表5：Computational Time obtained using TS, SA and TP.TS (100-500node)(sec)

Algorithm	100	200	300	400	500
TS	7.25	93.66	423.36	1357.88	2210.26
SA	29.49	75.41	147.51	230.50	307.12
TP.TS	3.50	24.51	90.8	254.34	573.07

表6：Computational Time obtained using TS, SA and TP.TS (600-1000node)(sec)

Algorithm	600	700	800	900	1000
TS	4456.19	5797.71	7385.28	9205.00	10832.89
SA	385.80	464.98	548.88	628.17	700.38
TP.TS	1068.53	1100.84	2059.99	2659.27	3543.63

対して探索を行うが，今回実験を行った4プロセッサ上では4通り組合せに対する探索しか行わないものであり，これによって計算時間の大幅な短縮を実現するとともに，有効な解を導出しているのが特徴である。また，この探索は各プロセッサ上のローカルループで局所的な探索が複数行われることで表3，4より解の質の改善が確認され，ノード数が少ないものを除いて，最悪値(MAX)でさえ Simulated Annealing を凌ぐ解が得られた。計算時間も Simulated Annealing には至らないものの，従来の逐次的な Tabu Search より大幅に時間が短縮され実用的な時間内で探索可能な効率的な並列アルゴリズムと言える。

9　おわりに

グラフ分割問題に対して基本的な構成からなる各種メタ戦略アルゴリズムを構成し数値実験によって実際的な評価を行った。その結果，Genetic Algorithm は本問題のような組合せ最適化問題に対しては良い結果が得られなかった。一方，Simulated Annealing，Tabu Search は優れた結果を示し，とくに Simulated Annealing においては解の質，計算時間ともに他のアルゴリズムを凌ぐ結果を示し，問題のサイズが大きいものに対しては計算時間の観点から Simulated Annealing が最も効果的であるといえる。以上の実際的な分析は応用分野における利用に際しての有効な知識となるであろう。

また，Tabu Search においてはその性質上，多大な計算時間を要した。そこで新たな並列というパラダイムを組み込んだ近傍分割型の並列 Tabu Search を提案し，Tabu Search において最も計算時間を要する近傍探索部分を各プロセッサに分散させることで効率的な並列化を行うことで計算時間を大幅に縮約することができた。この手法は問題サイズにあわせてプロセッサ数を増加させることで，計算時間をさらに短縮することも可能であり，より大きな問題でも現実的な時間で解を得ることができるであろう。さらに，Token Passing を用いた問題分割型の並列 Tabu Search を提案し，このアルゴリズム

によって計算時間の大幅な短縮を可能とし通常のSimulated Annealingを凌ぐ解を得ることができた。この手法ではTokenによって並列計算環境に依存せずに部分問題を割り当てることを可能とした資源管理を考案し，プロセッサ数やプロセッサ性能が異なるものが混在するシステムにおいてもできるだけ効率的に解を得ることを可能とした。また，各プロセッサには完全に独立した部分問題が割り当てられるため，プロセッサ独自の戦略で部分問題を解くこともでき，一つの問題に対して様々な戦略を複数同時に適用することも実現できる。同時に各プロセッサにおいて他のプロセッサの状態に依存せずに非同期に探索を行うこともできるであろう。このようにシステムに対して柔軟な対応を実現したことで，ワークステーションクラスタ特有のスケーラビリティを生かしたアルゴリズムであるといえる。

参 考 文 献

[1] 室田一雄：離散構造とアルゴリズム IV，近代科学社，1995。

[2] 斎藤，三十尾，志澤，丸川，秋山，野口，鬼塚：分子動力学法プログラム AMBER と Barnes-Hut tree code の並列化による高速化，情報処理学会論文誌，Vol.40，No.5，pp2142-2151，1999.

[3] MPI：メッセージ通信インターフェイス標準（日本語訳ドラフト），http://www.ppc.nec.co.jppi-jndex.html, 1996.

[4] 藤沢克樹，久保幹雄，森戸晋：Tabu Search のグラフ分割問題への適用と実験的解析，T.IEEE JAPAN, Vol.114-C, No.4, pp.430-437，1994.

[5] Roberto Battiti, Giampietro Tecchiolli: Parallel biased search for combinatorial optimization: genetic algorithms and TABU, Microprocessors and Microsystems Vol.16 No.7, pp.351-367，1992.

[6] SAUL A.KRAVITS, ROB A.RUTENBAR: Placement by Simulated Annealing on a Multiprocessor, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, VOL.CAD-6，NO.4，pp.534-549，1987.

[7] C.-N.Fiechter: A parallel tabu search algorithm for large traveling salesman problem, Discrete Applied Mathematics 51，pp243-267，1994.

[8] Michel Toulouse, Teodor G. Crainic, Michel Gendreau: Communication Issues in Designing Cooperative Multi-Thread Parallel Searches, Meta-heuristics: theory and applications, Kluwer Academic Publishers, 1996.

[9] 荒木，加地，山本，鈴木，大内：遺伝的アルゴリズムによる最適系列分割問題の解法，電気学会論文誌C分冊，Vol.120-C，No.2，pp.215-221，2000.

[10] Emile Aarts, Jan Korst: Simulated Annealing and Boltzmann Machines, JOHN WILEY&SONS, 1989.