数　値　解　析　37－9
（１９９１．７．１７）

トランスピュータネットワークにおける 0-1ナップサック問題の分枝限定解法

梶田哲史　渡辺敏正　翁長健治

広島大学工学部
７２４　東広島市鏡山１丁目４番１号

本稿の目的は，0-1ナップサック問題に対する疎結合並列計算機環境下での並列分枝限定解法において，節点探索法の違いや変数選択法の違いが計算時間，加速率等にいかなる影響を与えるかを調べることである．即ち，0-1ナップサック最大化問題をメッシュ型結合のトランスピュータネットワーク上で並列計算する場合の総計算時間及び加速率が，(1) 分枝節点の探索ルール，(2) 分枝変数の選択ルール，等によってどのような影響を受けるかを詳細に調べることを目的としている．

本稿では，分枝節点探索ルールとしては従来からの(1) 上界値最大節点を選択するBest upper-bound 選択，(2) 深さ最大節点を選択するDepth-first 選択，を用いる．分枝変数選択ルールとしては，(i) 子節点の上界値極大の変数選択，(ii) ＬＰ最適解を利用したピボット変数選択，(iii) 単価最大の未固定変数選択，なる既存の３つのルールに加えて，[13]で提案されている (iv) より大きな下界値を生成する変数選択，なる選択ルールを採用し，これらの探索／分枝ルールの組合せの下での実験結果に基づき解析を行う．また，探索節点選択ルールによっては得意，不得意なデータパターンが存在すると考えられるので，問題のパターンと探索ルールの関係についても解析をする．

Parallel Branch-and-Bound Algorithms for solving 0-1 Knapsack Problems on a Transputer Network

Satoshi Kajita, Toshimasa Watanabe and Kenji Onaga

Faculty of Engineering, Hiroshima University,
4-1, Kagamiyama 1 chome, Higashi-Hiroshima, 724 Japan

The subject of the paper is to analyze how node-search and variable-selection strategies affect computation time and speedup of parallel branch-and-bound algorithms for solving 0-1 knapsack problems on a mesh-connected transputer network.

There are two kinds of choice in a branch-and-bound algorithm: node(subproblem)-serching; variable selection. We adopt the following two rules of node-searching: (1) Best upper-bound rule; (2) Depth first rule. As variable-selection rules, there are three rules existing; (i) variables giving a maximal upper bound, (ii) pivoting-variables in LP relaxation, (iii) maximum unit-cost variables. In addition we consider the following rule which is proposed in our previous paper: (iv) variables producing a better lower bound. Experimental analyses with respect to these rules of node-searching / variable-selection are given.

1. まえがき

整数計画法等の組合せ最適化問題はＮＰ-困難であり最適解を求めるためには，最悪の場合，網羅的な探索を必要とする．組合せ最適化問題の一般的解法として分枝限定法が古くから研究されている[1,2,3,4]．分枝限定法の基本構造（分枝操作，限定操作，上下界値計算，探索法等）について現在の知見が[5]に集大成されていて，我々が本稿で使用する概念や基本性質はこの本に基づいている．

分枝限定法は一般解法であるが故に，問題の規模が大きくなると膨大な計算時間が必要とされる場合があるので，計算時間短縮を主な目的としてその並列化に種々の試みと解析がなされている[6,7,8,9,10,11]．[9]は最大化問題の最良上界値探索において，加速度＝（１台のプロセッサによる処理時間）/（ｐ台のプロセッサによる処理時間）がp以上となる現象，異常加速は，限定操作に用いる上界値g_Lが分枝木の根を含む上位部分においてすべて最適値f*に等しい場合にのみ発生することを示した．[10]は共有メモリ結合多プロセッサシステムを用いた並列分枝計算の加速効果を，ランダムに生成した分枝木についてシミュレートし，異常加速が発生することを示した．また，[11]はトーラス結合型多プロセッサシステムの負荷的分散効果を調べた．[12,14]ではトランスピュータを使った並列アルゴリズムの研究が行われている．

分枝限定法には，分枝木のどの節点で分枝を行うか（分枝節点探索），問題中のどの変数を固定するか（固定変数選択）の二つの選択があり，これらの選択方法が分枝限定法の並列化に与える影響を明らかにしてゆくことは，関連研究における一つの指針を与えるという意義を持つものと思われる．

本研究は，疎結合並列計算機環境の下での０-１ナップサック問題の並列分枝限定解法において，節点探索法の違いや変数の選択の違いが分枝木の探索にいかなる影響を与えるかを調べるものである．即ち，０-1ナップサックの最大化問題をトランスピュータネットワークにより並列計算する場合の総計算時間及び加速度が

1. 分枝節点の探索ルール
2. 分枝変数の選択ルール

等によってどのような影響を受けるかを詳細に調べる事を目的としている．

本稿では，分枝節点の探索ルールとしては従来からの(1)上界値最大節点を選択するBest Upper-bound探索，(2)深さ最大節点を選択するDepth-First探索，を用いる．分枝変数の選択ルールとしては(i)子節点中の上界値極大の変数選択，(ii)ＬＰ（線形計画法）最適解を利用したピボット変数選択，(iii)単価最大の未固定変数選択，なる既存選択ルールに加えて，[13]で提案された(iv)より大きな下界値を生成する変数選択，なる選択ルールを採用し，これらの節点探索，変数選択ルールの組合せの下での実験結果を示す．また，探索節点選択ルールによっては得意，不得意なデータパターンが存在すると考えられるので，問題のパターンと探索ルールの関係についても言及する．関連結果として，[13]において，２台のプロセッサに対するシミュレーションによる実験結果を報告している．本稿は４台のトランスピュータによる実験結果の報告である．

2. 0-1ナップサック問題

2.1. 0-1ナップサック問題の分枝限定解法

$1 \le j \le n$のとき，品物jの重量をb_j，その価値をa_jとしたとき，総重量が制限W以下で総価値f(X)を最大にするようにn種類の品物を選択する問題をナップサック問題と呼ぶ．特に各品物の個数が最大１個に限定させる場合を０-１ナップサック問題といい，以下のように定式化される．

$$\text{P: maximize } f(X) = \sum_{i=1}^{n} a_i x_i$$

$$\text{subject to } \sum_{i=1}^{n} b_i x_i \le W$$

$$x_i = 0 \text{ or } 1; \quad a_i, b_i = \text{正整数}.$$

制約式が複数個ある場合は，多次元0-1ナップサック問題と呼ぶ．

0-1ナップサック問題はＮＰ-困難であることが知られており，通常次のような分枝限定法により解かれる．以下[5]に従って簡単に解法を説明する．原問題Pに対し制約$x_1=0$または１を付加すると(n-1)変数の子問題$P(x_1=0), P(x_1=1)$が生成される．更に制約$x_2=0$または１を付加したときの(n-2)変数部分問題を$P(x_1 = 0, x_2 = 0), P(x_1 = 0, x_2 = 1), P(x_1 = 1, x_2 = 0), P(x_1 = 1, x_2 = 1)$と書くことにする．例えば$x_1, x_2, \ldots, x_n$の順序で変数を制約する時の親子関係は図１の２分木で表わされ分枝木と呼ばれている．許容解を持たない部分問題は分枝しないことにすれば，分枝木の葉は一つの許容解に対応し，その深さの最大値は変数の個数nに等しい．内部節点に対応する部分問題の変数固定は根から同節点に至る木道のラベルを並べたもので与えられるので，許容解は根Pからその葉に至る木道のラベルを順に並べたものである．変数固定$J \subseteq \{1,\ldots,n\}$を持つ部分問題P(XJ)は次式で定式化される:

$$\text{P(XJ): maximize } f(X) = \sum_{i \notin J} a_i x_i$$

$$\text{subject to } \sum_{i \notin J} b_i x_i \le W$$

$$x_i = 0 \text{ or } 1 (i \notin J). \quad (\text{各} x_j (j \in J) \text{は0または1に固定済み．})$$

P(XJ)の最適値をf*(XJ)と書き，その上界値，下界値を計算する関数g_U, g_Lを導入する：

$$g_L(XJ) \le f^*(XJ) \le g_U(XJ).$$

上界値$g_U(XJ)$を計算する一つの方法は，未固定変数の0-1整数制約を$0 \le x_i \le 1$なる実数制約に緩和したＬＰ問題$P^{LP}(XJ)$を解き，そのときの最適解を$f^*_{LP}(XJ)$と書き，$g_U(XJ)=f^*_{LP}(XJ)$とおくことである．まず以下の(i)(ii)を満たす様に添字を付け換える：

(i)未固定変数は$y_1, y_2, \ldots, y_m$，固定変数は$y_{m+1}, \ldots, y_n$（つまり，$J=\{m+1,\ldots,n\}$）である．

(ii) $\dfrac{a_1}{b_1} \ge \dfrac{a_2}{b_2} \ge \ldots \ge \dfrac{a_m}{b_m}$

このとき，

$$P^{LP}(XJ): \text{ maximize } f(y) = \sum_{i=1}^{m} a_i y_i + \sum_{j \notin J} a_j y_j$$

$$\text{subject to } \sum_{i=1}^{m} b_i y_i \le W' \left(= W - \sum_{j \notin J} b_j y_j\right)$$

と書ける．ここで，

$$\sum_{j=1}^{q-1} a_j \le W \text{ 且つ } \sum_{j=1}^{q} a_j > W$$

なるq ($1 \le q \le m$)を求めて

$$x^* = (x^*_1, \ldots, x^*_q, \ldots, x^*_m, x^*_{m+1}, \ldots, x^*_n)$$

とおく．但し，

$$x^*_1 = \ldots = x^*_{q-1} = 1,$$

$$x^*_q = \frac{W' - \sum_{i=1}^{q-1} b_i}{b_q},$$

$$x^*_{q+1} = \ldots = x^*_n = 0.$$

このとき，ＬＰ最適解は

$$f^*_{LP}(XJ) = \sum_{i=1}^{q-1} a_i x_i^* + a_q \frac{\left(W - \sum_{i=1}^{q-1} b_i\right)}{b_q} + \sum_{j \in J} a_j x_j^*.$$

なお，比a_i / b_iは品物 i の単位重量あたりの価値（単価と呼ぶ）を表わしており，(ii)の順序は単価降順の変数順序と呼ばれる．$f^*_{LP}(XJ)$がP(XJ) の許容解であるならば$f^*(XJ)=f^*_{LP}(XJ)$である．

下界値$g_L(XJ)$を計算するにはP(XJ)の一つの許容解X_0を見つけ出す必要がある．最も簡単な方法は品物番号順に重量制限に達するまで品物をナップサックに詰める事である．ただし品物iを詰込むと重量制限をオーバする場合には飛越しを行うことにする．例えば重量列(5, 3, 10, 4, 6, 2)と制限 W=15 が与えられたとき，まず先頭の5, 3を取る．次の10を取ると総重量＝18となって制限W=15をオーバするので飛越し，次の4を取る．6を飛越して2を取る．このようにしてバック許容解と呼ばれる一つの許容解X_0^{PACK}= (1,1,0,1,0,1) が得られる．$g_L(XJ) = f(X_0^{PACK})$とおく． ＬＰ問題$P^{LP}(XJ)$が許容解を持つとき$g_L(XJ) = f^*(XJ)$である．

分枝限定法による解法手順を以下にまとめておく．なお，分枝節点探索ルール，分枝変数選択ルールは次節2.2で説明する．

《0-1ナップサック問題の分枝限定解法》

1. 先端ノード集合A ← {P} （Pは原問題），固定変数集合J ← φ，暫定値z ← $g_L(X\phi)$，最適解←φと初期化する．
2. A = φなら計算終了．A ≠ φなら分枝節点探索ルールに従ってAの中から部分問題P(XJ)を選ぶ．P←P(XJ).
3. 下界値$g_L(XJ)$を計算する．$g_L(XJ)$ > 暫定値 zならば，z ← $g_L(XJ)$，最適解←{ X_0^{PACK} }とする．ただし，X_0^{PACK}は下界値 $g_L(XJ)=f(X_0^{PACK})$ を算出したPのバック許容解．
4. LP 問題P^{LP}を解き，その解x*より上界値$g_U(XJ)=f^*_{LP}(XJ)$を計算する．$g_U(XJ) < z$であるか又はx*がPの許容解であるならば，ステップ6へ行く．
5. 分枝木の節点Pにおいて分枝変数選択ルールに従って変数x_jを選び，二つの子問題$P(X_j=0), P(X_j=1)$を生成し，J←J ∪{j}, A ← A ∪ {$P(X_j=0)$, $P(X_j=1)$}とする．
6. A← A−{P}とし，ステップ２へ戻る．

2.2 分枝節点探索ルールNSRと分枝変数選択ルールVSR

先端節点集合（既生成の節点でまだテストが加えられていないものの集合）Aの中から分枝節点を探索ルール(NSR)によって選出し，未固定変数の中から分枝変数選択ルール(VSR)によって，新たに0または1に値を固定する変数を選択する．多数のルールがこれまでに提案されており，特性と性能の基本的な部分はほとんど固まっているようであり，[5]に集大成されている．NSR は，次の(1)(2)が主流である．

(1)Best Upper-bound Node Selectionルール(BUN)：先端節点集合Aの中で上界値が最大である節点を選択する．複数候補があるときは深さの大きい節点を選択する方式(BUN-DFNルール)か，或いは下界値の大きい節点を選択する方式(BUN-BLNルール)が考えられる．生成される部分問題をヒープに維持する．

(2)Depth-First Node Selection ルール(DFN)：先端節点集合Aの中で深さの大きい節点を選択する．複数候補があるとき上界値が大きい節点を選択する方式(DFN-BUNルール)か，或いは下界値が大きい節点を選択する方式(DFN-BLNルール)が考えられる．生成される部分問題をスタックに維持する．

BUN は展開しなければならない節点総数を最小にする点で最良の探索法と云われている．しかし，メモリーに保持しなければならない先端節点集合Aの大きさが最悪の場合深さdと共に指数オーダーで増大する．DFN ルールは探索に偏りがあり，最適解のない分枝に誘導される危険性が高いが，先端節点集合Aの大きさは常に変数nの一次オーダーに維持されるという利点をもつ．

分枝節点が選定されると，次は未固定変数中で，どの変数を固定するかを決定しなければならない．VSRについては，次の(i)-(iii)が既知である．

(i)子節点の上界値を極大にする変数を選ぶ．

(ii) LP問題$P^{LP}(XJ)$の解$x^*=(x^*_1,...,x^*_q,...,x^*_n)$において $0<x^*_q<1$の場合は変数x_qを選ぶ．(これをピボット変数と呼ぶ．)

(iii)未固定変数のうち単価の大きな変数を選ぶ．

これらに加えて，[13]ではVSRとして新たに次のルールが提案されている．

(iv)Better Lower-bound Variable Selectionルール (BLV)：より大きな下界値を生成する変数を選定する．

BLVは実は上記(ii)(iii)を合体して一般化したものであると考えられる．単価降順に並べた変数を用いたバック許容解により下界値$g_L(XJ) = f(X_0^{PACK})$を定義したが，さらにより良い下界値を得るためには，適当なk ≤nに対して１次，２次，...，k次のバック許容解X_1^{PACK}, X_2^{PACK},..., X_k^{PACK}まで求めその最大値を選ぶ．ここで，原バック許容解X_0^{PACK}において値0を取る変数を左側から(k+1)個$x_{c0}, x_{c1},..., x_{ck}$とし，$x_{ci}$を1に固定した上でバック許容解を求めたものを$X_i^{PACK}$としている．

$\hat{g}_L(XJ) = \max\{f(X_0^{PACK}), f(X_1^{PACK}),..., f(X_k^{PACK})\} = f(X_r^{PACK})$

但し，$x_{cr} = X_r^{PACK}$の固定変数

とおけば，新しい下界値$\hat{g}_L$はg_Lよりも改良されている．BLVでは節点P(XJ)から分枝木を展開するためにx_{cr}を0と1に固定する．k次であることを明示するために k-th BLVと呼ぶこともある．

3. トランスピュータネットワーク上での分枝限定解法

分枝限定法の並列アルゴリズムはアーキテクチャにより２つに大別される．１つは，共有メモリ型マルチプロセッサシステム上での並列アルゴリズムで，プロセス間通信に共有変数を用いる．従って，部分問題の転送や暫定解の更新，転送が高速に行えるという利点を持つ．その一方で，問題の維持に大きなメモリ領域を必要とし，またプロセッサ台数の増加に伴いメモリのアクセス競合が起こり、処理効率の低下を引き起こす．もう１つは，分散メモリ型マルチプロセッサシステム上でのアルゴリズムであり，前者と比べてプロセッサ台数の増加による処理効率の低下は少ない．しかしメッセージ転送にはチャネルを介した１対１通信を用いるので，チャネルが直接つながっていないプロセッサへのデータ転送には中間に位置するプロセッサでの中継が必要となり，転送によるオーバーヘッド増加が考えられる．

3.1. トランスピュータネットワーク

トランスピュータはINMOS社が開発した32bitマイクロプロセッサであり，他プロセッサとの通信機能として４本のシリアルリンクを持っている．シリアルリンクを結合することによってトランスピュータネットワークを構成することができ，また，プロセス記述言語としてOCCAM2を使用している．トランスピュータは，１台のプロセッサ上で複数プロセスを並列実行することができるように内部にスケジューラが用意されており，トランスピュータ内でのマルチタスキングが可能である．またプロセスを複数のトランスピュータに分割することも可能である．本稿での並列アルゴリズムはトランスピュータネット

ワーク上で実行されるものであり，分散メモリ型非同期式である．

3.2. 並列分枝限定アルゴリズム

トランスピュータネットワーク上での並列アルゴリズムについて説明する．アルゴリズムは4つのプロセスから成っている．これらは1つのトランスピュータ内のタスクを処理すると共に，トランスピュータ間の通信，すなわち負荷分散を行う．

3.2.1. トランスピュータ内のプロセス　トランスピュータネットワークでは各プロセッサは独立に動作する非同期式である．このため，複数プロセッサからの通信入力がある度にその処理が開始できるように入力部は計算部と独立している必要がある．また，トランスピュータ間の双方向通信を可能にするために出力部も独立している必要がある．更に，入力部で受けとったメッセージを直ちにスタックや出力部に送ることにするので計算部とスタック部も独立させる．よって，以下に示す合計4つのプロセスがトランスピュータ内で実行される：

stack()	スタックを管理する．
input()	隣接プロセッサからの入力を管理する．
output()	隣接プロセッサへの出力を管理する．
calculation()	計算，終了の判断を管理する．

また，プロセッサ間通信に使用する主なメッセージは以下の通りである．なお，全プロセッサの状態を管理するためにrootプロセッサを設ける．以下で述べる，idleメッセージ，busyメッセージはrootプロセッサへ各プロセッサの状態を知らせるためのメッセージである．

request：	隣接プロセッサに部分問題を要求する．
revival：	部分問題を受け取ったことを隣接プロセッサに知らせる．
empty：	スタック／ヒープが空であることを示す．
idle：	部分問題がないことをrootプロセッサに知らせる．
busy：	部分問題を受け取ったことをrootプロセッサに知らせる．
stop：	プロセッサを停止させる．

3.2.2. 負荷分散手法　各プロセッサは，受け取った部分問題を根とする部分分枝木をスタック／ヒープの中で個別に維持する．スタック／ヒープが空になった場合には隣接プロセッサから部分問題を転送してもらい，負荷分散を行う．本稿での負荷分散手法は以下の通りである．

(i)プロセッサpは自分のスタックが空になるとすべての隣接プロセッサp'にrequestメッセージを送る．（図2）

(ii)pは，隣接プロセッサp'から部分問題が送られてくれば，p'以外に出したrequestを消すために（部分問題を受け取ったことを知らせる）revivalメッセージをp'以外の全隣接プロセッサに送る（図3）．

(iii)pは隣接プロセッサp'からのrequestメッセージを受け取った時，スタック／ヒープ中に部分問題が2個以上あればスタックの1番上にある部分問題をp'に送る．1個以下ならば，requestメッセージを記憶しておき2個以上になった時にp'に送る．

(iv)pはp'からのrevivalメッセージを受け取れば，記憶していたp'からのrequestメッセージを削除する．

(v)pは自分のスタック／ヒープが空でないときは，スタック／ヒープの1番上の部分問題を解く．また，生成された部分問題をスタック／ヒープに入れる．

(vi)pは隣接プロセッサp'からの暫定解の更新値を受け取り且つそれにより暫定解の更新が起これば，p'以外の全隣接プロセッサにその更新値を送る．

3.3. 加速率と使用効率

並列計算における加速率A(p)はプロセッサをp台使用する時の計算時間T(p)が一台の時の計算時間T(1)に比べて何倍小さくなるかを示す量であり，

$$A(p) = T(1) / T(p)$$

で与えられる．プロセッサ使用効率η(p)はp台の各プロセッサが使用される時間比率を示し

$$\eta(p) = A(p) / p$$

で与えられる．各プロセッサは100％を超えて働くことは出来ないと直感的には考えられるが，[10]は[5]の手法を用いてランダムに生成した分枝木に対する分枝限定計算シミュレーションによって，異常加速 $A(p) > p$ が発生しうることを示した．並列分枝限定法の加速率に関して[5],[9],[10]の結果が[13]にまとめられている．

BUNの欠点は先端節点集合Aが最悪の場合に指数的に増大する危険があることである．その欠点を除去するのがDFNであり，p台のプロセッサが管理する先端節点集合の和集合の大きさはpnを超えない．一般的に採用されているルールはDFNである．DFNの欠点は偏った探索に誘導されると，それを脱するのに長い時間を要することである．複数台のプロセッサによる並列探索によって早期により良い暫定解が生成され脱出時間を大幅に短縮出来る，というのが異常加速発生の理由である．同様な理由で異常減速，$T(p)>>T(1)$，も起こりうる．即ちDFNはプロセッサ使用効率 $\eta(p)=T(1)/(T(p)*p)$ のぶれが大きすぎるのが欠点である．分枝ノードが定まった時，どの変数で分枝を行うかを決定しなければならない．未固定変数中で，1に固定するとより良い下界値を発生する変数を選び，これによって分枝すればより良い暫定値をより速く生成され，より良い枝刈が期待出来る．この分枝変数選択ルールをBetter lower-bound variable selection(BLV)と呼ぶ．分枝節点探索は深さを優先し，対が生じた時下界値の大きいものを採用するルールをDFN-BLNルールと呼ぶ．DFNの欠点を除去する可能性のある新しいルールとして[13]が提案したものである．

4. 実験

4.1. ナップサック問題の生成

実験に用いるナップサック問題は，$1 \le i \le n$ に対して，以下の各変数 a_i, b_i, W を次の(i)~(iii)により発生することにより定められる．

(i) 4つの変数 f_{min}, f_{max}, b_{min}, b_{max} に任意の値を与え，
　単価 $\varepsilon_i = a_i/b_i \in [f_{min}, f_{max}]$，及び
　重量 $b_i \in [b_{min}, b_{max}]$
　を計算機によりランダムに発生させる．

(ii) $a_i = f_i * b_i$ とする．

(iii) $0<W_p<1$ なる W_p を任意に与え，
　重量制限 $W \in [(W_p - 1/20)\sum_{i=1}^{n} b_i, (W_p + 1/20)\sum_{i=1}^{n} b_i]$
　をランダムに生成する．

また，重量列のパターンとして

1) 増減がランダムな重量列
2) ランダム単調増加重量列
3) ランダム単調減少重量列

を考え，これらを単価降順列と組み合わせて問題を生成する．

4.2. 実験結果と解析

前節で生成された問題に対して，2つの分枝節点探索ルールと4

つの分枝変数選択ルールをそれぞれ組み合せて実験を行い，実験結果に基づき解析を行う．4台のトランスピュータを環状接続したネットワーク上で実験した．（将来的にはメッシュ結合ネットワークを目標としているが，現状では4台のため環状である．）

4.2.1. 節点探索法／変数選択法による比較　以下の条件の下で実験結果を表1，2に示す．

(1) $f_{min}=b_{min}=10$, $f_{max}=b_{max}\in\{100, 200\}$,

(2) $W_p\in\{0.4, 0.6, 0.8\}$,

(3) n=40, 80, 120, 160, 200, 400,

(4) 増減がランダムな重量列,

に対するDFN1－4，及びBUN1－4の実験結果である．表1は計算時間，表2は加速率を示す．なお，n=40,80,120,120,160,200 については，$f_{max}=b_{max}=200$とし，W_pを0.4, 0.6, 0.8の中からランダムに一つ選んで計算を5回反復した平均計算時間である．n=400については，$f_{max}=b_{max}=100$とし，$W_p=0.8$と固定して計算を5回反復した平均計算時間である．図4，5に表1のPAR: DFN1－4及びPAR: BUN1－4のグラフを，また図6，7に表2のそれらを示す．実験結果から明らかになったことは，

(a) 深さ優先探索が最良上界値探索よりも総計算時間が短いこと，

(b) 固定変数選択ルールにおいては，(ii) ピボット変数選択ルール，　或は(iii) 単価最大の未固定変数選択ルールが計算時間が短いこと

である．このことは上述の重量列の選び方にかかわらず成り立っている．但し，その選び方により(ii)の方が(iii)よりも計算時間が短い場合とその逆の場合がある．なお，紙面の都合で図，表等は省略するが，n=600, 700 に対してもn=400の場合と同様な条件下で，DFN3に対する実験結果も得ている．例えば，n=600のときには SEQ: 28.33(s), PAR: 1903.5(s); n=700 のときには SEQ: 5.68(s), PAR:87.98(s)である．但し，SEQの方が計算時間が大幅に短くなっており，この原因については更に検討する必要がある．

4.2.2. 重量列による影響　前述の重量列の選び方が計算時間等に与える影響を調べるため

(1) $f_{min}=b_{min}=10$, $f_{max}=b_{max}=110$,

(2) $W_p=0.4, 0.6, 0.8$,

(3) n=40, 60, 80, 100,

に対するDFN1－4，及びBUN1－4に関して実験した．なお，W_pの各々の値に対する1回の計算時間のみを計測した．

＊単調増加重量列－単価降順列：n=100に対するDFN1－4，及びBUN1－4の実験結果をそれぞれ図8，9に示す．この組み合せでは，展開する分枝木の左側に最適解が存在する問題が生成されやすいが，効率の良い（つまり単価の高い）変数が0に固定される傾向があるとも考えられる．よってピボット変数を次の分枝節点変数に選び分枝を行うと早期によりよい暫定解が見つかり，より多くの節点の枝刈りが起こる可能性がある．即ち，DFN–(ピボット変数選択ルール)の組み合せが，総計算時間が短くなることが予想される．

＊単調減少増加重量列－単価降順列：n=100に対するDFN1－4，及びBUN1－4の実験結果をそれぞれ図10，11に示す．この組み合せでは，分枝木の右側に最適解が存在する問題を生成する可能性がある．よって，単価の大きいものから順に変数固定を行えば最適解が存在しないような分枝節点への探索が起りにくくなると考えられる．

4.2.3. W_pによる影響　$W_p=0.4, 0.6, 0.8$ の各々の場合に関しては，図4－7に示している．W_pによる影響は，今回の実験からは特に明らかにはできなかった．更に詳細な解析が必要と思われる．実験データからは，W_pの変化が総計算時間に与える影響はDFNが最も少ないという結果が得られている．

5. あとがき

本稿では，0-1ナップサック問題をトランスピュータネットワーク上で解くための並列分枝限定法において，分枝節点探索／固定変数選択ルールが計算時間等の解探索に与える影響を解析することを目的とした実験結果の一部を報告した．実験結果から，深さ優先探索が最良上界値探索よりも総計算時間が短く，固定変数選択ルールにおいては，(ii) ピボット変数選択ルール，(iii) 単価最大の未固定変数選択ルールが計算時間が短いという結果が得られた．トランスピュータの台数を増やしてメッシュ結合ネットワーク上でn=400以上の場合に対する実験を充実させる予定である．

参考文献

[1] Land,A.H and Doing,A.G.:An automatic method for solving dicrete programming problems,Econometrica 28, pp.497-520 (1960).

[2] Little,D.C., Murty,K.G., Sweeney,D.W., and Karel,C.:A algorithm for thetraveling salesman problem, Operatiopn Research, 11, pp.978-989 (1963).

[3] Balas,E.: An additive algorithm for solving linear programs with zero-one variable, Operations Research, 13 pp.517-545 (1965).

[4] Geoffrion,A.: Integer programming by implicit enumeration and Balas's method, SIAM Review, 7, pp.178-190 (1967).

[5] 茨木俊秀：組合せ最適化－分枝限定法を中心として（講座，数理計画法8），産業図書(1980).

[6] El-Dessouki, and Huen,W.:Distributed enumeration on network computer, IEEE Trans. Computer C-29, pp.818-825 (1980).

[7] Harris,J and Smith,D.: Hierachical multiprocessor organizations, Proc. 4th Ann. Symp. Computer Architecture, pp.41-48 (1977).

[8] Wah.B. and Ma.Y.: MANIP - A parallel computer system for implementing branch-and bound algorithm, Proc. 8th Ann. Symp. Computer Architecture, pp.239-262 (1982).

[9] T.Lai and S.Sahni,: Anormalies in Parallel Branch-and Bound Algorithms, Communication of ACM ,pp.27-06 (1984).

[10]今井，吉田，福村：" 分枝限定アルゴリズムの並列化とその評価"，信学論(D), J62-D,6, pp.403-410 (1976-06).

[11]川口，真栄田：" トーラス型マルチプロセッサシステム上での分枝限定アルゴリズム"，信学論(D), J72-D-I No4 pp.254-261 (1989-04).

[12]Oliver Vornberger: "Lord Balancing in a Network of Transputers, Distributed algorithms", Lecture Note of Computer Science 312, Springer Verlag, pp.116-126, (1987).

[13]末広，渡辺，翁長：" 並列0-1ナップサック分枝限定法における節点，変数選択効果" 信学技報，CPSY 89-49, pp.25-31 (1989-08).

[14]R.Luling and B.Monien: " Two strategies for solving the Vertex Cover Problem on Transputer Network", Lecture Note on Computer Science 392 (1989).

[15]M.J.Quinn: "Analysis and Implementation of Branch-and-Bound Algorithms on a Hypercube Multicomputer", IEEE Trans. Computers, Vol.39, No.3 ,pp.384-387 (1990-03).

	1	2	3	4	5	6	7
ε	27.0	19.0	18.0	17.0	16.0	15.0	14.0
b	7	10	12	14	16	18	20
a	189	190	216	238	256	270	280

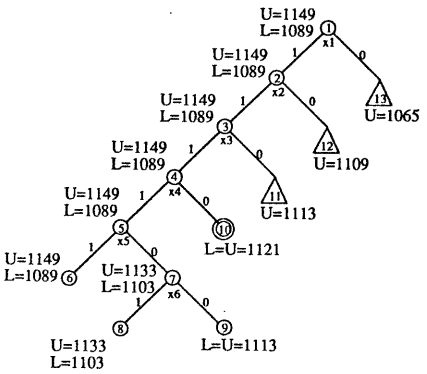

図1．分枝木の例．

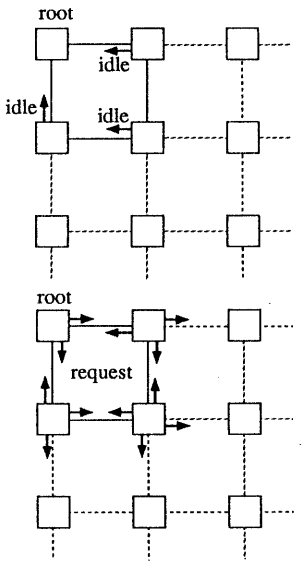

図2．request メッセージ．

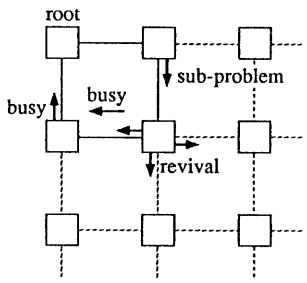

図3．revival メッセージ．

表1．実験結果（計算時間）の一部．DFN1－4はそれぞれ固定変数選択ルールに対応する．BUN1－4についても同様である．SEQ, PAR は各々1台，4台のトランスピュータによる直列処理，並列処理の計算時間（単位：秒）である．

	N	40	80	120	160	200	400
SEQ	DFN1	0.630	3.670	19.087	19.084	13.724	295.336
	DFN2	0.203	0.844	3.211	3.619	5.643	116.039
	DFN3	0.217	0.815	2.642	4.954	5.613	26.925
	DFN4	1.463	9.559	91.922	82.245	83.997	1673.190
	BUN1	0.185	3.540	15.656	21.393	17.558	1.423
	BUN2	0.230	0.979	2.178	6.038	5.030	1.271
	BUN3	0.265	1.064	2.402	5.288	4.662	41.444
	BUN4	1.226	6.681	14.596	43.032	54.393	2.650
PAR	DFN1	0.206	0.671	4.270	5.200	5.840	98.623
	DFN2	0.094	0.346	0.720	1.160	1.650	24.910
	DFN3	0.120	0.420	1.110	1.920	2.210	13.727
	DFN4	0.628	2.731	22.530	33.030	36.050	222.443
	BUN1	0.228	0.812	4.210	5.010	5.060	29.877
	BUN2	0.103	0.351	0.900	1.740	1.870	2.639
	BUN3	0.147	0.502	1.530	2.260	2.590	12.842
	BUN4	0.468	2.296	9.540	22.530	16.840	107.093

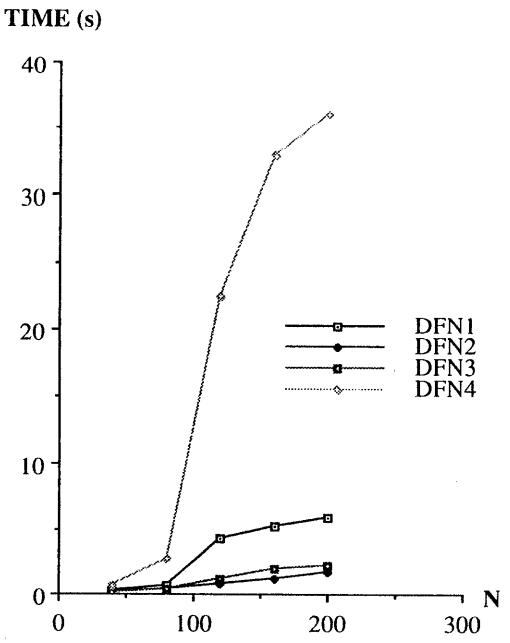

図4．表1の計算時間 PAR: DFN1－4 のグラフ．

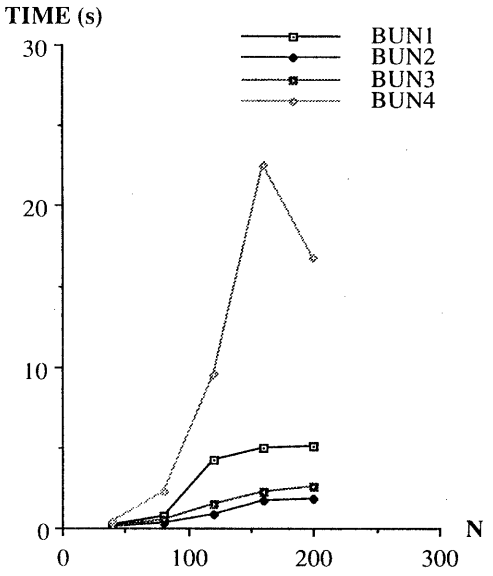

図5．表1の計算時間 PAR:BUN1－4のグラフ．

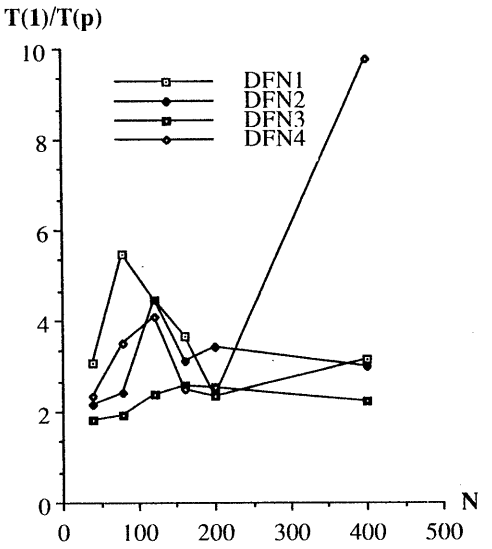

図6．表2の加速率 PAR: DFN1－4のグラフ．

表2．表1に対応する加速率．

N	40	80	120	160	200	400
DFN1	3.06	5.47	4.47	3.67	2.35	3.15
DFN2	2.16	2.44	4.46	3.12	3.42	2.99
DFN3	1.81	1.94	2.38	2.58	2.54	2.25
DFN4	2.33	3.50	4.08	2.49	2.33	9.75
BUN1	3.46	4.36	3.80	4.27	3.47	0.05
BUN2	2.23	2.79	2.42	3.47	2.69	0.40
BUN3	1.80	2.12	1.57	2.34	1.80	3.34
BUN4	2.62	2.91	1.53	1.91	3.23	0.07

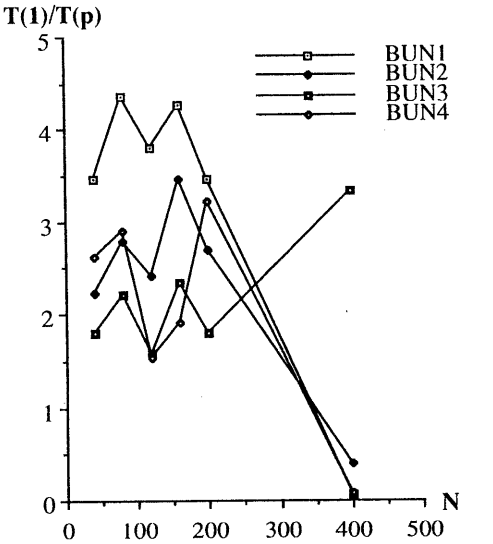

図7．表2の加速率 PAR:BUN1－4のグラフ．

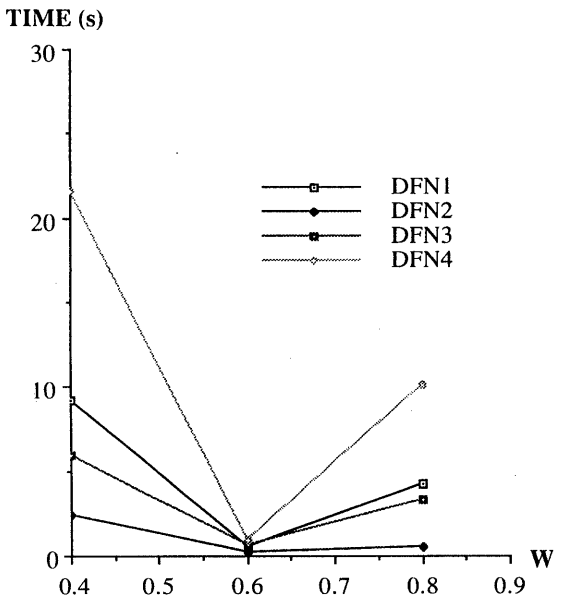

図８．単調増加重量列に対するPAR: DFN1－4の結果(n=100).

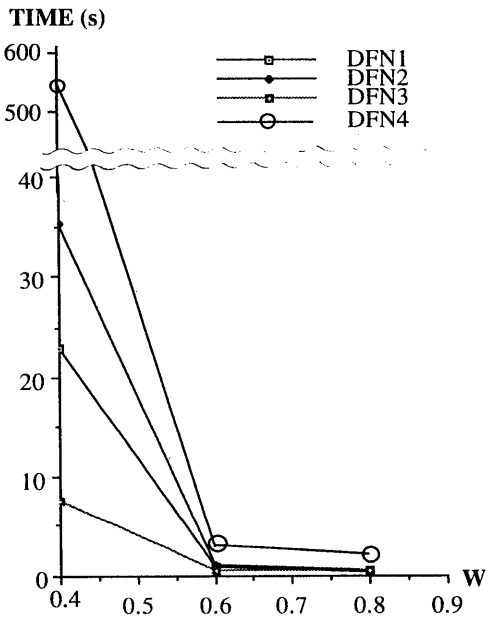

図１０．単調減少重量列に対するPAR: DFN1－4の結果(n=100).

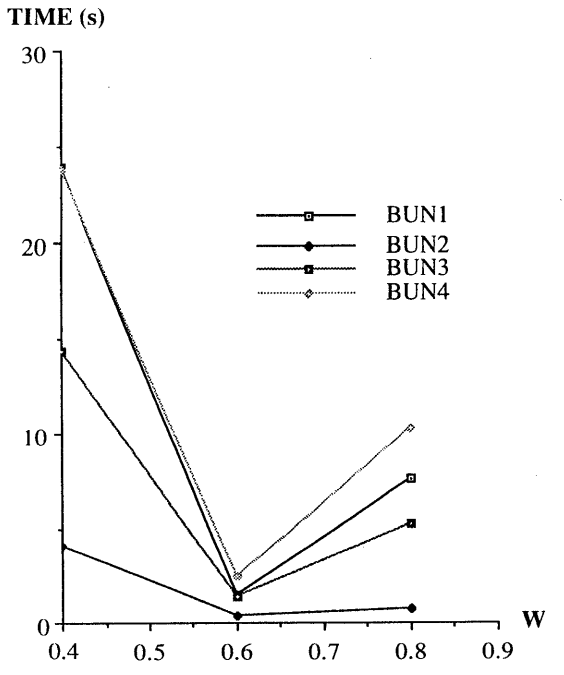

図９．単調増加重量列に対するPAR:BUN1－4の結果 (n=100).

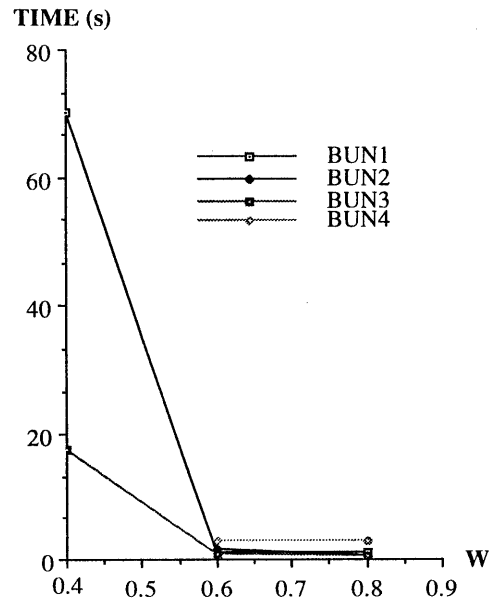

図１１．単調減少重量列に対するPAR:BUN1－4の結果 (n=100).