分枝限定法を用いた部分和問題
－振込金仕訳プログラムへの摘用－

Subset sum problem by using branch-and-bound method
－Application to the program of itemizing amount paid on account－

森 下 伊三男
Isao Morishita

要 旨

本学では教職員が立て替え払いで購入した種々の物品代金及び旅費交通費は、後日まとめて経理課より振り込まれることになっている。そして、振り込まれた金額の内訳について知るには、その都度経理課への問い合わせが必要となっている。しかし、多くの教職員がその問い合わせをする事は経理課の事務量を増加させることになり、あまり好ましい事とは言えない。そこで、筆者は振り込まれた金額の内訳を明らかにする振込金仕訳プログラムの開発を試みた。この仕訳に関する問題は、いわゆる部分和問題と同じであり、情報科学の分野では NP-困難な問題としてよく知られている。本稿では、より高速な手法を試み、その実験結果、併せて、プログラムについて報告する。

1. はじめに

一般に、n 個の正整数 $a_1, a_2, \ldots, a_n$ 及び正整数 b が与えられたとき、$a_1, a_2, \ldots, a_n$ の中から適当に選ばれた値の和が b となるようにできるかどうかを判定し、できる場合にはその組み合わせを求める問題は部分和問題といわれている[1, 2]。更に、それぞれの a_i に対して c_i という量を対応させ、その c_i の和が最大になるような一般化された問題はナップザック問題と呼ばれている。これらの問題は、多項式オーダーの計算量では解けないことが知られており、NP-困難な問題となっている。しかし、NP-困難な問題であっても、この問題に関するアルゴリズムは存在し、たとえば、すべての組み合わせを計算し、その中から解を見つける事によって解く事が可能である。ただし、これには指数関数オーダーの計算量が必要とされる。しかし、現実の問題として、n が小さければ現在のパソコン環境でも解を求める為の計算は十分に実行可能であり、更に何らかの条件が付けば比較的簡単に解を求めることができる場合もある。

本稿では、特に部分和問題を取り上げ、キーの値となる b（以下、「キー値」と呼ぶ事にする）を構成する要素の候補となるすべての正整数 a_i に対して d_i という属性を与えることにする。そし

て、キー値に対する属性 d_b を判断基準にして、各 d_i を調べることで最初に部分和問題の対象となる構成要素候補を抽出し、正整数 a_i の要素数を減らすことにより問題のサイズの縮小を考えた。次に、分枝限定法を応用し、解となり得ない組み合わせを事前に排除する事によって、無駄な組み合わせの計算を行うことを防ぐ試みをした。分枝限定法とは、系統的に解の候補を列挙し探索すると同時に、最適になり得ないと前もって知ることのできる解を上手に“刈り取り”無駄な探索をしないようにする、という考え方である[3, 4, 5]。この考え方は、根付き木 T(A)を用意し、根 r にはすべての入力($a_1, a_2, \ldots, a_n \mid b$)を対応させ、r の左の子には要素 a_n を選ばないもの($a_1, a_2, \ldots, a_{n-1} \mid b$)を対応させ、r の右の子には($a_1, a_2, \ldots, a_{n-1} \mid b-a_n$)を対応させる。こうして2分木を再帰的に作っていく。各ノード v に対応する($a_1, a_2, \ldots, a_i \mid b'$)が決まったら、$b' > 0$ である限り、v の左の子に($a_1, a_2, \ldots, a_{i-1} \mid b'$)を対応させ、v の右の子には($a_1, a_2, \ldots, a_{i-1} \mid b'-a_i$)を対応させる。このようにして、$i = 0$ となるか $b' \leqq 0$ となるまで繰り返してノードを作っていき、最終的にできる木が T(A)となる。この木 T(A)の中で $b' = 0$ となっているノードへのパスが部分和問題の解に対応し、右の子におりたものだけを集めれば、それが解を形成する要素ということになる。しかし、解が存在しない場合、ノードの数は $2^{n+1}-1$ となり、例えば一つのノードで 4 バイトのメモリを使うとすると、$n = 30$ の場合には約 8 GB のメモリが必要となる。従って、実行時間もさることながら必要となるメモリ資源も無視できない事になり、パソコンでの実行は現実的に困難ということになる。

そこで、本稿では、実行時間は若干犠牲となるものの、再帰手続きにより各ノードを一つ一つ調べながらスタックに積み込んでいくような形で根付き木 T(A)を調べることにした。つまり、初めに T(A)全体を作り、その後で $b' = 0$ となるノードを探すのではなく、T(A)を作りながらノードを調べ、解となる可能性のないノード以下はすべて廃棄していく事にする。これによってスタックのためのメモリは必要となるが、木 T(A)の全体をメモリに作りあげること無く解を見つけだす事が可能となる。更に、複数の解がある場合、最初に見つかった $b' = 0$ のノードのみを解として受け入れることにして実行時間の短縮を図ることにする。これに対し、初めに T(A)を形成しておけば、すべての存在するノードを調べることで、すべての解を得ることができる。しかし、本稿で扱う振込金仕訳プログラムの場合、適当な条件を与えることで、実際問題として複数の解が存在するケースは少なくなる事から、複数の解が存在する場合には最初に見つかった解のみを得ることに限定して議論を進めることにする。基本的なアルゴリズムとしては、あらゆる組み合わせを試みるべくプログラムを考える必要があり、プログラム開発環境 Delphi を用いてプログラムを作成した。また、その効果についての実験も試みた。

本稿では、以下に、まずプログラムの中心となる部分を紹介し、次に、実験の為に用意したデータを利用して検索効果の調査をし、その結果を報告する。

2. 解を求めるためのプログラム

部分和を求め、与えられたキー値に等しくなるような要素の組み合わせを求めるプログラムの中心となる部分をリスト1に示す。リスト1はプログラムの一部であり、実装している同名の手続きとは内容が若干異なっており、省略してある部分もある。しかし、それらは他の手続きとの関連等で必要とされる部分であり、ここでの議論には本質的に関係しない部分である。以下にプログラムの説明をする。

```
procedure Make_TempSum(tsum, ist : integer);
{データチェックルーチン：再帰呼び出し：中心部分}
{tsum, ist：部分和、チェックするデータの開始番号}
var
   n : integer;
begin
   for n := ist to imax do
      begin
         tsum := tsum + data[n];
         hit_no[n] := true;
(* 和が total_amount より大きくなったの場合は data[n] を引いて戻る   *)
(* 和が total_amount 以下の場合のみ、等しいか否かチェックし、及び *)
(* 次の data[n+1] を加えるように再帰呼び出しをする                  *)
         if total_amount >= tsum then
         begin
            find := total_amount = tsum;
            if find then begin Finish; Exit end;
            Make_TempSum(tsum, n+1);
            if find then Exit;
         end;
         tsum := tsum - data[n];
         hit_no[n] := false;
      end;
end;
```

リスト 1

リスト1の中の data[n]、 hit_no[n]は手続き外で宣言された配列で、前者は部分和を構成する要素の候補となる値が入る配列、後者はそれが解を構成する要素であるか否かの値が入る配列である。また、 total_amount も同様に手続き外で宣言された変数であり、部分和に対するキー値が手続きに入る前に入れられている。 tsum 、 ist はリスト内の注釈行にあるように、それまでの部分和及び配列 data の中で部分和に加えられるべき要素の最初の添え字の値である。なお、最後の添え字の値は手続き外で宣言され、手続きが呼ばれる前に imax に与えられている。 Make_TempSum が再帰呼び出しされる手続き名で、部分和が等しくなるまで次々と再帰呼び出

しを行って、要素を tsum に加えていき、tsum が total_amount より大きくなったら最後に加えた要素を除外し、次の要素を加えて同様に再帰呼び出しを繰り返すようになっている。もし、途中で部分和が与えられた値に等しくなった場合はフラグとして手続き外で宣言された論理型変数 find を true にし、別に定義された手続き Finish で結果を出力し、その後、これまでに再帰呼び出しでスタックに積み込まれたすべての再帰状態を順に戻って手続きを完了することになっている。

3. 振込金仕訳プログラムへの適用

ここでは、上のプログラムを適用する具体的な例として、教職員が研究の為に立て替え払いをした複数の支払及び出張旅費に対して振り込まれた金額について、その内訳を調べる問題を取り上げる。従って、上記の n 個の正整数 $a_1, a_2, \ldots, a_n$ に対応する値が教職員の立て替え金額及び出張旅費であり、正整数 b に対応するキー値が経理課からの振り込み金額となる。この問題の解は経理課が知っており、経理課からの振り込みに対して、どの立て替え払いが対応するか、その内訳を経理課に問い合わせれば済む事である。しかし、現実問題として、振り込みの度に毎回その様な問い合わせを経理課にしていては経理課本来の業務が圧迫されるため、現実的な対応とは言えないことになる。そこで、容易に操作できる小さなサイズ（例えばフロッピーディスク・ベースで稼働する程度）のプログラムを作成し、この問題に対応させることを試みた。併せて、部分和問題における実際の計算量についての実験も同じプログラムで行った。

立て替え払いのデータの場合、伝票には領収書などの年月日が記載されている。出張旅費の場合は、本学に戻ってから出張旅費精算書兼復命書を提出するため、出張期間末日の翌日が伝票の場合の日付に相当する事が多い。これらの伝票あるいは精算書（以下、伝票で代表させる）が実際に経理課に届き、経理処理が終った後に振り込み処理が行われることになる。つまり、早く処理が進んだとしても、振り込み日より数日前までに切った伝票が振り込みの対象となる。従って、この時間的な流れを考慮することによって、振り込みの対象となる伝票が限定され得ることになる。ただし、経理処理が始まるのは、伝票に記載され日付ではなく、実際に経理課に伝票が届いた日以降であるため、必ずしも日付順に振り込みがなされるとは限らない。例えば何葉かの伝票を保持しており、後にまとめて提出する場合には、その中のどれが先に処理されるかは定かではない。そのため日付順から大きくはずれて振り込みがなされる可能性もある。しかし、大局的には、古いの日付の伝票ほど振り込みが先に行われる可能性は高いといって良い。

そこで、振り込み金額から対応する立て替え払いの伝票抽出の為に、以下のような定式化を行うことにした。必要最小限の入力データとして、立て替え払いをした伝票の金額 $a_1, a_2, \ldots, a_n$、それらに対応する日付 $d_1, d_2, \ldots, d_n$、求めるべき部分和を与えるキー値として振り込まれた金

額 b 及び日付 d_b を考えることにする。ここで、日付を条件として、問題の対象となる伝票金額を抽出し、その後に部分和問題を解くということは、

$$\sum_{i \in I} a_i = b \ , \ a_i \in A \ , \quad A = \{ \ a_i \mid d_i \leqq d_b \ , \ i = 1, 2, \ldots, n \}$$

となる i の組（集合 I）を求めることになる。ここで、A は d_b で与えられた日付より遅くない日付 d_i を持った a_i からなる集合であり、その要素 a_i のある組み合わせの和が b に等しくなるような i の組（集合 I）が部分和問題の解となる。実際問題としては、先に述べた伝票処理の時間的な流れから考えて、条件式の中の等号は除いても問題とはならない。この段階での伝票金額に対するこの操作は、単に部分和問題の対象となるデータの数を限定していることであり、本質的には、日付の条件が満たされているデータのみを部分和問題の対象としていることに等しい。

一方、こうして選ばれた集合 A の要素 a_i の中には b より大きな値を持った要素が存在する可能性もある。その場合、その要素についても集合 A から除外できる。以下では、日付けの条件が満たされており、かつ、値が b 以下であるような要素のみが集合 A の中に入っていることを前提とする。

4. 最悪の場合の実行時間

ここでは、前節で述べたような前提条件の下で、最悪な場合の実行時間について考える。図1に、すべての組み合わせを試みたときの実行時間を示す。すなわち、単純に集合 A の要素が n 個である場合、2^n 通りの組み合わせすべてをチェックする場合である。たとえば、どんな組み合わせの部分和を作っても b にならない場合、つまり解が存在しない場合には、それを確かめるにはすべての組み合わせを調べる必要があり、最悪の場合の実行時間ということになる。ただし、b = 0 の場合は考えないことにすれば、すべての要素を除いた場合、すなわち部分和がゼロの場合は除かれることになり 2^n-1 通りの組み合わせということになる。図1では、■が DEC のノート型パソコン VP735（CPU：Intel MMX Pentium 233 MHz、memory：96 MB）、●が Epson Endeavor のデスクトップパソコン VZ-4000（CPU：Intel Pentium III 500 MHz、memory：256 MB）を用いた n = 32 までの計測結果である。開発環境は共に Borland Delphi 5（OS：Windows 98）である。両者の実行時間の比はほぼ一定で VP735（■）の方が約 1.6 倍の時間を要している。この比は CPU のクロック周波数の比 2.14 より小さく、パソコンのハードウェア環境及び Windows 環境の違いと考えられる。例えば、バックグラウンドで実行されているプログラムについては VP735 では 8 本、VZ-4000 では 22 本であった。これらの条件を等しくしても CPU 周辺のハードウェア環境が異なるため、比は 1.8 程度にしかならなかった。ここでは、パソコンの

性能調査をすることが目的ではないのでこれ以上のチェックは行わなかった。環境の違いによる両者の実行時間の比はともかく、いずれのパソコンの場合でも問題のサイズとなるデータ項数 n が大きくなるに従って指数関数的に実行時間が増加しているのが分かる。

図1．データ項数 対 実行時間

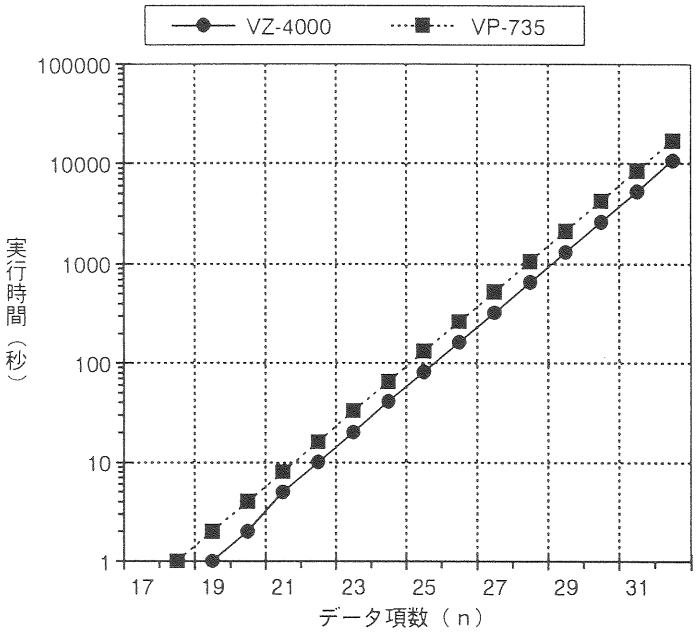

図1で示されている実行時間は解が存在せず、すべての組み合わせを計算するのに必要となる時間（最悪の場合の時間）であるが、もし解が存在する場合には平均としてその半分の時間で解が得られることが予想される。さらに、今問題としている立て替え払いの伝票の場合には、時間的に早い伝票が処理される可能性が高く、それらの組み合わせを優先的に調べていくことで更に実行時間が短縮できることが予想できる。そこで、より一般的なデータの場合の実行時間を調べるために、乱数値を用いた実験をおこなった。その結果について次に述べる。

5. 乱数値による実験

伝票のように時間的な流れの情報が存在せず、データに何ら順序づけが無いような一般的な場合の実行時間を求める為に、 n＝24 の場合について、次のような実験を試みた。まず、解を構成する要素の候補 a_i の値として 2^i (i= 0 , 1 , 2 . . . , 23)を用意した。次に、キー値データとなる b の値として、 $2^{23} < b < 2^{24}$ を満たす乱数を 30 個用意し、それぞれについて解が求まるまでの実行時間を計測した。ここで用意したキー値の場合、必ず解が存在し、平均で 12 項の構成要素を持つ解の実際的な検索実行時間を求めることができる。また、候補要素のデータ a_i の並びとして昇順あるいは降順の場合、更に各候補要素のデータに3種類の異なる二桁以下の乱数

列(乱数列－1，乱数列－2、乱数列－3)を振り、それぞれの乱数列について、それぞれ降順及び昇順となるようにデータを並べ直した場合についても実験を行った。参考までに実験に用いた候補要素データ及び乱数列を表1に示す。ただしこの表は候補要素データについて昇順に並べた場合の表である。また、表2に検索のキー値として用いた 30 個の b の値を小さい順に示す。なお、この実験には、図1の計測で用いた Epson　Endeavor の VZ-4000 を利用した。

表１．実験用データ（２４項）

No.	データ	乱数列-1	乱数列-2	乱数列-3
1	1	71	34	54
2	2	46	91	48
3	4	27	75	66
4	8	67	53	19
5	16	65	54	0
6	32	40	68	30
7	64	59	77	31
8	128	62	62	79
9	256	91	5	64
10	512	61	20	24
11	1024	77	21	69
12	2048	95	50	39
13	4096	39	41	61
14	8192	66	65	58
15	16384	81	51	23
16	32768	32	43	63
17	65536	96	47	73
18	131072	75	38	9
19	262144	69	1	21
20	524288	7	28	53
21	1048576	98	26	36
22	2097152	86	15	41
23	4194304	88	3	94
24	8388608	15	63	1

実験結果を表3に示す。この実験では、候補要素データの性格上、データを降順に並べた場合は一瞬(1秒以下、実験した時の感覚では数分の1秒)で解が求まった。しかし、昇順に並べた場合には表3にあるように平均 17.4 秒を要する事が分かった。データを降順にした場合には、大きい値から順に部分和に組み込まれるため、刈り取りの効果が大きく作用したと考えられる。これに対し、昇順にした場合、再帰呼び出しが深くならないと刈り取りの効果が出ないため、表の中では最も大きな実行時間を必要とする結果となった。表3には、30 個の乱数に対する実行時間の平均値に加え、その標準偏差、最大値、最小値を示した。ここで、 0 とあるのは、1秒未満の場合を意味する。各乱数列でソートした場合の実行時間はいずれの場合も昇順の方が小さかった。乱数列についての昇順、降順に対する実行時間の違いについては、偶然の結果であろうと思われる。更に、条件を変えた実験が必要となるかもしれないがここでは省略した。

表２．検索に用いたデータｂ

8523369	10709016	13759287
8681929	10853127	13805570
8807256	10892709	14156698
8827001	10985051	14821564
8940972	11855825	14958525
9268523	11902468	15260368
9696994	12679720	15686471
9767924	12781490	15780180
9923176	12881788	16675752
10657171	13088426	16736966

実験で用いたすべてのキー値 b は候補要素の最大値

2^{23} より大きく、解の中に必ずその最大値を持つ要素が含まれている。もし、キー値 b が 1 以上で 2^{24} 未満の乱数値であるならば、より実行時間は小さくなることが期待できる。なぜなら、候補となる要素を抽出する段階で b より大きな値を持った要素はすべて除外できるからである。つまり、部分和問題のサイズを縮小することができるからである。この実験では、サイズの縮小を避け、同じサイズでの部分和問題とする為にあえて $2^{23}<b<2^{24}$ という条件を b に付けた。その点では本稿が想定している振込金仕訳の問題より負荷がより大きくなっている。データ数 24 項の場合の最悪の時間は図1にあるように約 40 秒(VZ-4000)である。従って、平均的には最悪の時間の半分となる約 20 秒と予想されが、乱数列に対する実験結果では実行時間は平均 13.1 秒、標準偏差 7.6 秒であった。これは、分枝限定法による刈り取りの効果によるものである。この刈り取りの部分をプログラムから除き、30 個の乱数データを同様に検索した場合の実行時間を表4に示す。この場合には、降順にソートしたデータについて瞬時に解が求まる場合は少なく、平均でも 10.4 秒を要している。また、乱数列に対する実行時間の平均値は 19.1 秒となり、理論通りに最悪の時間のほぼ半分になっている。従って、分枝限定法による刈り込みの効果によって約 3 割程実行時間が短くなったことになる。更に、振込金仕訳の場合には日付の条件が加わり、より短い時間で解が求まる可能性が高いことになる。更に、データが降順に並んでいれば実行時間はより小さくなることが予想できる。

表３．乱数値に対する実行時間（秒）

	データ	乱数列-１		乱数列-２		乱数列-３	
	昇順	昇順	降順	昇順	降順	昇順	降順
平均値	17.4	12.7	15.2	11.9	16.0	9.5	13.2
標準偏差	10.4	6.2	8.0	8.1	8.2	7.3	6.1
最大値	38	21	30	24	36	23	22
最小値	2	0	1	0	2	0	1

表４．分枝限定法を用いない場合の実行時間（秒）

	データ		乱数列-１		乱数列-２		乱数列-３	
	昇順	降順	昇順	降順	昇順	降順	昇順	降順
平均値	22.2	10.4	20.2	21.3	17.6	21.3	13.8	20.3
標準偏差	11.5	5.9	8.6	10.9	11.0	9.4	9.8	11.6
最大値	37	18	30	38	36	36	28	39
最小値	2	0	3	2	0	2	0	1

一方、表2で示した 30 個のデータに対する解の構成要素数は平均で 12 項、標準偏差は 2.3 項、最大 16 項、最小 8 項であった。ただし、データはランダムに並んでいるため、構成要素の数が少ないからといって必ずしも短い時間で解が求まるということではない。従って、解の要素数については特に実行時間に対する影響はここでは議論できない。ただし、もし、解がすべての要素からなる場合には、上記のアルゴリズムの場合、再帰呼び出しを最後のデータまで順にスタックしていくだけの極めて短い時間で解が求まる事になる。また、実際のプログラムでは、候補要素の総合計がキー値よりも小さいときは明らかに解が存在しないので、リスト1で示した手続きを呼び出す前に総合計をチェックし、もし解が存在しないことが明らかな場合は、解の無いことを示すメッセージを表示する様にしてあることを付け加えておく。

6. プログラムの紹介

図2. 振込金仕訳プログラムのウィンドウ

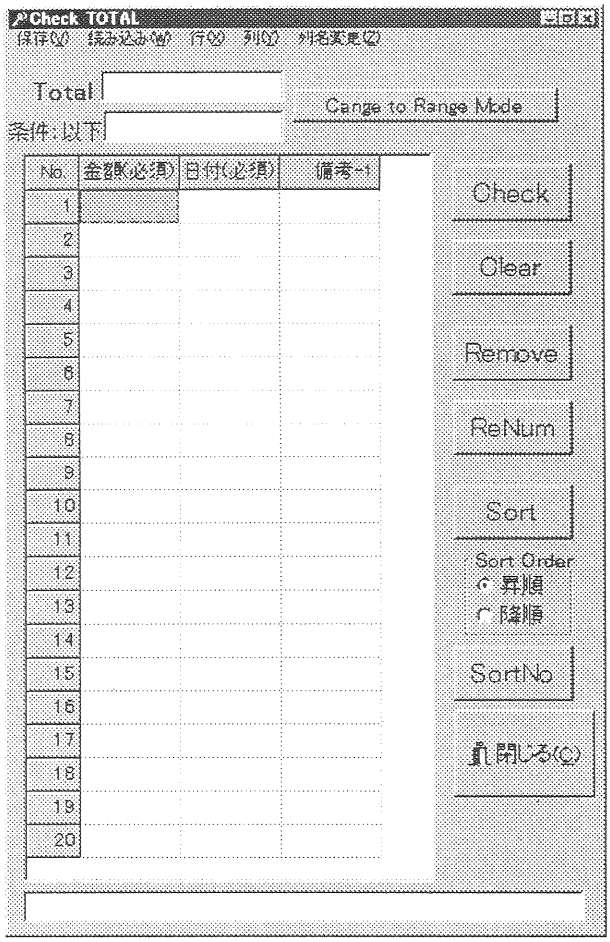

図2に実際に構築したプログラムのウィンドウを示す。メニューの[保存]、[読み込み]は候補要素のデータの保存、読み込み、[行]、[列]は下にある表の行又は列の挿入・削除、[列名変更]は表の最上行にある列タイトルの変更を行う為のコマンドである。「金額」の列に立て替え払い金額を、「日付」の列に伝票の日付を入力する。仕訳をするには、「 Total 」の欄に振り込まれた金額を入力し「 Check 」ボタンをクリックする。このとき、「条件:以下」の欄に入力された日付より新しい日付を持つデータは除外し、それより古い日付(日付の値としてはより小さな値)を持つデータのみが検索の対象となる。この欄に何も入力がない場合はすべてのデータが検索の対象となる。また、この条件は第2列目に入力された値に適用され、図2では「日付」となっているが、日付のデータに限定せず、数値データであれば特に制限は設けられてはいない。従って、一般的には、第2列目のデータが「条件:以下」の欄に入力された値以下の行が検索の対象となる。求められた解は、その構成要素となる金額が入力されている行の色を変えることで表示される。あるいは、一番下にある欄に最左列の[No.]の値が表示される。「 Clear 」ボタンは得られた解の行の色表示を消す。また、「 Remove 」ボタンは解として得られた色の付いた行を削除する。このボタンあるいはメニューのコマンドで行を削除した場合、[No.]の値は変更されずにそのまま保持される。その為、再度ナンバリングし、最上行から順に 1,2, ... と番号を振るには「 ReNum 」のボタンをクリックする。「 Sort 」ボタンは、カーソルのある列について下のオプションボタンで指定された順(昇順または降順)にソートを行う。また「 SortNo 」は、同様に[No.]について昇順又は降順にソートを行う。ソートする場合は、左側の2列については数値と

してソートし、それ以外の列（備考-1 及びそれより右側に挿入した列）については文字としてソートされる。最後に「閉じる」ボタンはプログラムの終了である。ここで用いた表は Delphi の StringGrid を利用しており、セルへの入力を行う編集モード（Edit Mode）とセルの範囲を指定してのカット・コピー＆ペーストの作業を行う範囲モード（Range Mode）の二つのモードを切り分けして利用する事になっている。従って、使い勝手は市販の表計算ソフトのような洗練されたインタフェイスとはなっていない。ここで、そのモード変更のためのボタンが「Change to Range Mode」である。Edit Mode の時には図のようになっており、Range Mode の場合には同じボタンが「Change to Edit Mode」と表示される。また、表の中で右クリックをすることによって、切り取り、コピー、貼り付け、消去のコマンドを実行できるようになっている。ただし、Windows 98 のクリップボードを利用したものではない為、このプログラム内に利用は限定されている。また、誤って「Check」ボタンをクリックし、長時間の検索に入ってしまった場合や検索実行を強制的に中止する場合を考え、「Check」ボタンをクリックした時には、そのボタンの脇に「Cancel」ボタンを表示し、実行を中止することができるようなっている。なお、このプログラムは EXE ファイル一つで実行可能であり、そのサイズは 426KB となっている。

7. まとめと今後の課題

ここでは、実際の立て替え払いデータを用いて実行を試み、今後の問題点を探った。筆者が平成11年度に立て替え払いした約 110 件の立替払い伝票の金額をデータとし、約 20 件の振込についてその振込金額を平成11年4月に遡って次の3通りの方法で仕訳する事を試みた。いずれの場合も、得られた解をその都度削除していき、古い順に振込金額の仕訳を行った。

（方法1）
入力は振込金額と振り込まれた日付とした。基本的には、日付で候補となる要素を抽出しているため、問題のサイズは小さく（平均で約5項）、解となった要素を順に削除していくことで、すべての場合でほとんど一瞬にして解が求まった。ただし、金額が同じで、伝票に記載された日付が近いケースで、日付の古い方を先に解と見なすという場合が1件あった。実際には、私の経理課への伝票提出の関係で、新しい日付の伝票に対する振込であったが、実験では、データの並びから古い方が解の要素となってしまった。これが、例えば、2葉の伝票の合計金額が他の1葉の伝票の金額に等しい場合には、次の（方法2）や（方法3）で述べるような問題が発生する可能性もある。しかし、近い日付でその様なケースが発生する可能性はあまり高くなく、また、実際にその様な問題が発生したとしても、個別に対応する事によって対処できる問題である。

（方法2）
振り込まれた日付についての条件を入力せず、単にデータを古い順または新しい順に並べた場

合の検索を試みた。その結果、解はすぐに求まるものの、本来の解とは別の解が求まるケースが発生した。かまわず解となったデータを削除していき、続けて次の振込金額に対する検索を行っていくと、古い順にソートした場合は8件目の振込金額の仕訳で解が見つからない状態に陥った。また、日付の新しい順にソートした場合では、9件目の仕訳で解が見つからなくなった。

(方法3)
立替払いの金額で降順又は昇順にソートし、(方法2)と同様に検索を行った。この場合も昇順では5件目、降順では9件目の仕訳で解が見つからなくなった。

(方法1)についてはほぼ問題なく実用に耐えうると判断できるが、(方法2)及び(方法3)については実用上の問題点が現れる。つまり、後の二つの方法では、候補となる要素が多いため複数の解が多く存在することが問題となる。従って、1年分の立て替え伝票を何ら条件を付けずにまとめて検索することは本プログラムでは問題が多いといわざるを得ない。ただし、通常は振り込まれた日付が情報として必ず得られているので、それで条件を付ければ問題の発生は極力抑える事が可能である。

最後に、今回作成したプログラムは十分に実用に耐えうると筆者は考えているが、なお、ユーザインタフェイスの部分では改良の余地が残されている。当面は、市販の表計算ソフトのような編集機能が充実した物にする必要があろう。また、例えば Microsoft の Excel の表からコピー＆ペーストでデータをインポートする事ができる様にすることも必要となる。これらの操作性向上に対する対応は今後の課題としたい。

参考文献

[1]浅野孝夫・今井浩「計算とアルゴリズム」、オーム社、pp.11-17 (2000)
[2]岩田茂樹「NP 完全問題入門」、共立出版、pp.1-24 (1995)
[3] Niklaus Wirth「Algorithm + Data Structures = Programs」、Prentice-Hall (1976)[邦訳: 片山卓也 訳「アルゴリズム ＋ データ ＝ プログラム」、日本コンピュータ協会、pp.176-180 (1979)]
[4]久保幹雄・松井知己「組合せ最適化[短編集]」、朝倉書店、pp.60-77 (1999)
[5][1]に同じ、pp.132-133

森下　伊三男　　(経営学部情報管理学科教授)