241 ALPS/Iでの箱入娘パズルの解法について

小林茂男 井田昌之 山方宏修 宮本聡
(青学大理工学部経営工学科) (理科大・理)

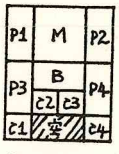
1. はじめに

筆者らが作成したLispマシンALPS/I[2](マイコンIntel8080に大容量メモリを付けたもので，Lisp処理系はインタプリタによる)は実用性を目的としているが，その機能を十分に果たすことをテストするプログラムの1つとして「箱入り娘パズル」[2]を取りあげてみた。これはちょうど，東大・後藤教授の『"箱入り娘パズル"程度の問題すら解けないようでは，実際に意味のある記号処理の問題を処理するシステムとしてはまったく迫力がない。よってここにLispシステムの性能試験用プログラムとして"箱入り娘パズルを解け"を提案したい』という御提案[3]に沿っている。そこで，この結果について報告したい。ところで，「箱入り娘パズル」はSliding-block puzzleの1つであり，図1に示されるように初期配置をとった10枚の駒をその枠組の中で空きを利用して，スライドさせて動かし，目的となる配置を得るパズルであり，M(娘)を，B(番頭)，P1～P4(父,母,下女,下男)，C1～C4(子供)の激しい妨害を排除しながら連れ出すという意味がある。また動かす際の1手とは，1つの駒に指をつけて動かすことのできる変化をすべて1手に数えるものとする。このパズル自体としては，すでに計算機によって解かれている。

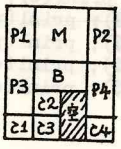
2. パズルの解法

このパズルの解法のアルゴリズムは基本的に後藤教授の文献[3]に従っている。解の探索法はbreadth-first法を用いており，解の最小手順を求めることができる。探索の手続きを記述した主関数nextconfは図2に示されている。考え方は，nを初期配置から出発しての手数とし，Sn1をn-1手目，Snはn手目で移行できる配置のすべての集合，dnはSn内のある1つの配置とするときに，Snの中に目的配置が表われるまで手数nを増やす。目的配置が見つかるとSn-1(注 変数名としてはSn1)中にあるはずの，次の1手で目的配置となる配置dn-1(注 変数名としてはdnですべて流用している)を見つける。次に，1手でそのdn-1になるdn-2を見つける。これを初期配置が見つかるまでくり返し，たどって戻ってくる。この過程において最小手順が求まる。nextconf中で用いている関数*Smoveは，ある1つの配置よから次の1手で得られる配置をリストにしたものを値に持つ関数であり，*fm2はALPS/Iのユーザが定義できるLispインタプリタとリンクのできるアセンブリ・レベルの関数名であり，これによって各駒の配置を1語に詰め，あるいは展開を行なっているが，使用にあたってはLispの範囲を超えないように吟味している。これは高速化のためのアセンブリ・レベルの関数使用である。m, b, p1～p4, c1～c4は各駒に対応した配置であり，global変数として定義されている。配置は

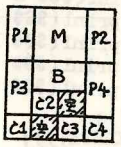
図1.



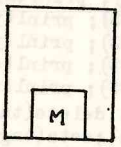
初期配置



1手配置(no1)



1手配置(no2)



目的配置

参考文献
1: 井田昌之 マイクロコンピュータを用いたLISPマシンALPS/I 情報処理(投稿中)
2: 小林慧 箱入り娘 数理科学 no115, Jan 1973
3: 後藤英一 連載Lisp入門 bit Vol.6 no13 1974
4: 井田昌之 LISPマシンALPS/Iの性能について 本予稿集 no214

数字に変換されて表わされている。（m=final の時が目的配置となっている）
この nextconf のアルゴリズムは，「箱入り娘パズル」のみでなく，他の問題にも適応できる。

3. 本解法の特徴

ALPS/I はマイコンを用いているために，この点に留意し，ALPS/I 上で処理しうるように refine し簡略化を行なっている。nextconf 自体もその意味で考えられており，新しい関数 Try[4] を用いて，目的とする配置が見つかった後の探索の途中打ち切りが行なえるようにしてあり，無駄を省いている。このような，1つだけが見つかれば，あとは不用という処理は多くあるので，簡略化と共に高速化のメリットもあるものと思われる。　また，集合の操作は，hash化配列[4]（2つの key（1方を配列名，他方を添字（添字は数に限らず，任意のシンボル又はリスト））によって定められるユニークな要素からなる集合）を用いてマーキングすることにより，メモリの消費を押え，かつ速度的にも有利とした。　関数 ※ smove による次の1手を決める手順は，解析形手順（論理的に解析して，次の1手を決める）と列挙形手順（経験的に次の1手を決める）との2つのアプローチがあるが，この2つのアルゴリズムの trade off を吟味して，$C_1 \sim C_4$ のみ解析形，他は列挙形の手続とした。　また，この手続は hash化配列に登録することにより，手続自体を高速に（associative）に取り出すことを可能としている。　さらに，鏡像の処置もこの中で行なっている。　1つの配置は1語(32bit)に収めることにより，ALPS/I では約1万1千通りの配置を一度に覚えることができる。　このため，外部ファイルの参照時間を少くすることが可能となる。

4. 謝辞

日頃御指導いただいている間野浩太郎教授に感謝いたします。

図2.

```
array(tag)
nextconf(n;snl) := prog((dn); prinl(n);  putspace(3);  print(length(sn));

      (null(sn) -> return(NIL);
       try(sn;λ((j);prog2(*fn2(0;j);eq(m;final))))
        -> progn( print(n);  print(KAERI);
                  prinl($$$ M=$); prinl(m);  prinl($$$ B=$); prinl(b);
                  prinl($$$ P1=$); prinl(p1);  prinl($$$ P2=$); prinl(p2);
                  prinl($$$ P3=$); prinl(p3);  prinl($$$ P4=$); prinl(p4);
                  prinl($$$ C1=$); prinl(c1);  prinl($$$ C2=$); prinl(c2);
                  prinl($$$ C3=$); prinl(c3);  prinl($$$ C4=$); prinl(c4);
                 return(try(snl;λ((k);try(*smove(k);λ((j);(*fn2(0;j);eq(m;final))))))))));
      mapc(sn;λ((j);seta(tag(j);ATTA)));
      value := nil;
      mapc(sn;λ((j);mapc(*smove(j);λ((k);(not(eq(tag(k);ATTA))
                                          -> value := cons(k;value))))));
      mapc(snl;λ((j);delete(tag(j))));
      (setq(dn;netconf(addl(n);sn;value))
        -> progn( *fn2(0;dn);
                  print(n);
                  prinl($$$ M=$); prinl(m);  prinl($$$ B=$); prinl(b);
                  prinl($$$ P1=$); prinl(p1);  prinl($$$ P2=$); prinl(p2);
                  prinl($$$ P3=$); prinl(p3);  prinl($$$ P4=$); prinl(p4);
                  prinl($$$ C1=$); prinl(c1);  prinl($$$ C2=$); prinl(c2);
                  prinl($$$ C3=$); prinl(c3);  prinl($$$ C4=$); prinl(c4);
                  mapc(sn;λ((j);delete(tag(j))));
                  mapc(snl;λ((j);seta(tag(j);DEKETA)));
                 return(try(*smove(dn);λ((j);eq(tag(j);DEKETA)))))))
```