Solving Nonlinear Block Stacking Problem by Operator Oriented Genetic Algorithm

MINAGAWA Masaaki

Abstract
To solve the nonlinear block stacking problem, this paper describes an approach based on an Operator Oriented Genetic Algorithm. The block states are represented in cellular coordinate space. Operators to generate block motions are given as production rules called basic-rules. A meta-rule is formed as a chain (an application sequence) of the basic-rules. Each meta-rule is an artificial chromosome for GA based search and its capability of goal attainment is tested. A collection of meta-rules forms the population of GA. Fitness value is given to each meta-rule and is calculated based on its rule length: the shorter the meta-rule length to attain the given goal, the higher the fitness value. That is, the problem is solved as an optimization (minimization of plan) problem. Based on the proposed methodology, experiments are carried out and some experimental results are shown.

1. INTRODUCTION

This paper reports a Genetic Algorithm[(4)] based approach to the well-known block stacking problem which is familiar in traditional AI field[(13)]. Informally, the problem can be stated as follows.

> Given an initial state and a goal state, find a motion sequence of the blocks to attain the goal.

In traditional approaches, means-ends analysis, hierarchical planning, least-commitment strategies etc. were proposed but resulted in limited success[(18)]. Due to the nonlinear nature (i.e conflict among block motions towards goal attainment[(14),(15)]) of the planning problem, any single strategy is hard to be effective.

The search space (the states of the given blocks, pre/post-conditions, operators) was represented in propositional form (e.g. ON(X, Y), ONTABLE(X)). In solving the problem, the initially given problem is divided into several sub-problems and, according to this, subgoals are generated. The search is carried out until each subgoal is completely attained.

Although the relevant block states are quite limited during a particular block

MINAGAWA Masaaki 札幌学院大学社会情報学部

motion, the search method requires keeping and searching costly huge state space.

Success of the search largely depends on description of the targeted problem as well as domain-dependent heuristics. Descriptions of block states, operators, conditions are not easy job. To obtain successful results, these have to be carefully described and heuristically tuned. Furthermore, the problem of describing relations between the result of operations and caused changes in the state space (i.e. the frame problem) is still unsolved.

To give analyses to such problems, recent papers show the result of computational complexity study on the block stacking problem(1),(2),(5),(7).

By describing the block stacking problem another way and viewing the problem as an optimization problem rather than theorem proving, we make a new approach using Genetic Algorithms (hereafter GAs). In the proposed approach, the states of blocks are represented using a cellular coordinate space, while traditionally the space was represented in propositional form (see Fig. 1). The use of cellular coordinate space eliminates necessities for delete/add-lists and can decrease the size of state space.

To date, some attempts were made to solve the problem using GA based approaches. Koza's approach (Genetic Programming paradigm(10)) is closely related to traditional AI methodology. S-expression of LISP code is represented as an artificial chromosome and LISP programs for moving blocks are searched. Koza hasn't shown general example in the literature(10). Hirano used another chromosome representation based on state transition graph(8). A state of blocks is represented as a node and feasible transition is represented as a directed edge of graph. Each numbered edge is coded as gene of the artificial chromosome. A minimum sequence of transitions is searched as a solution. In Hirano's approach, generation of the graph may become crucial problem.

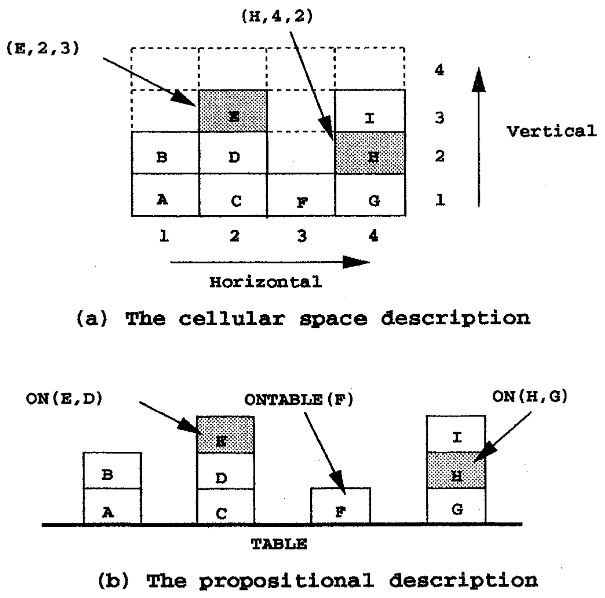

Figure 1 The state descriptions of the blocks world

One major obstacle in finding plans for the block stacking problem is that many-to-many mapping among block state transition exists(19). Before moving a block, the decision have to be made on "move which block to which position". To consider this aspect of the problem, in the proposed approach, a primitive block motion is given by a production rule which is triggered according to the given current block state(11),(12). To produce a series of motion, a chain of the production rule is formed. When fired, each block moves according to the given sequence of the rule. The set of the sequences forms the population of GA.

This is similar to the Classifier System (CS[9]) which is a machine learning system based on string-formed production rules (classifiers), the bucket-brigade algorithm, and Genetic Algorithms. Evaluation criteria, however, is different. Evaluation is given to a series of rules while the CS gives evaluation to each rule. In other words, we attempt to find solutions without using the bucket-brigade mechanism which assign higher credit to frequently fired rules.

Fitness (environmental reward) values are associated with meta-rule (MR) length. The length is given as the number of effective basic-rules (BRs) to attain the goal state. The shorter the MR length, the higher the fitness. This can be rephrased as: solve the problem as minimization of MR length attempting to obtain the plan having smallest number of block motions. We use variable-length string[3],[6] to permit redundant representation of solutions. To solve similar stack example, successful sub-strings are inserted to solve the larger size problems effectively.

In this problem setting, information given at each locus is sequentially interpreted and a plan (phenotype) is obtained. Each gene represents an operator while, in most GA applications for function optimization problem, it represents pure numerical value. This is called "operator oriented GA" and was firstly applied to the sliding block puzzle problem by Suh[16],[17]. In Suh's example, operators were motions of blank position of the puzzle (e.g. UP, DOWN, RIGHT, LEFT moves).

2. THE PROBLEM DESCRIPTION

We describe the problem P using the following 7-tuples.

$$P=(Q,\Sigma,\delta,E,q_0,q_G,S)$$

where

- Q: the finite set of states
- Σ: Finite set of input symbols
- δ: State transition function
- E: evaluation in the environment
- q_0: Initial state ($q_0 \in Q$)
- q_G: goal state ($q_G \in Q$)
- S: the set of input symbol string

A state q ($\in Q$) is describes as

$$q=\{(b,h,v)\}$$

- b: a block ($b \in B$, B: the set of blocks)
- h: horizontal grid position ($h \in I$)
- v: vertical grid position ($v \in I$)
- I: the set of integers ($I > 0$)

The state transition function δ maps a state q_i to another state q_j ($q_i, q_j \in Q$).

$$\delta\ (q_i,s)\ = q_j$$

where

$$S=\{s\}$$

$$s=\sigma_1\sigma_2\cdots\cdots\sigma_i\cdots\cdots\sigma_n\ (i \in I, \sigma_i \in \Sigma)$$

The input symbols are sequentially interpreted from left to right side. By interpreting these input symbols, the state transition occurs step by step (block by block).

The problems is, therefore, to find the set of input symbol strings which terminates the state transition by reaching the given goal:

$$\textit{Find}\ S=\{s \mid \delta\ (q_0,s)=q_G\}$$

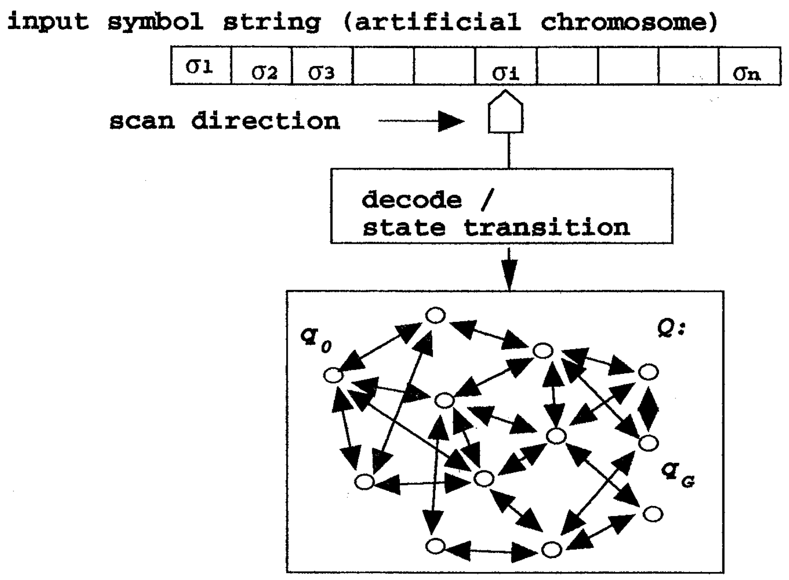
input symbol string (artificial chromosome)
σ1
σ2
σ3
σi
σn
scan direction
decode / state transition
q0
Q:
qG

Figure 2 The finite state automaton using the operator oriented genetic algorithm

Based on the descriptions above, we attempt to construct a finite state automaton based on the operator oriented GA (Fig. 2).

3. STATE SPACE DESCRIPTION

The state space described in propositional form is translated as follows;

ONTABLE (X) :

$\exists (h,v)$ *such that*

$assign\ (h,v) = X,\ and\ v=1 \qquad (h,v \in I)$

ON (X,Y) :

$for\ u_i=(X,h_i,v_i) \in q,\ u_j=(Y,h_j,v_j) \in q,$

$h_i=h_j,\ and\ v_i=v_j+1$

CLEAR (X):

$\exists (h,v)$ *such that*

$assign\ (h,v)=X,\ and$

$assign\ (h,v+1)=\phi$

The function *assign* returns a block name or null set (ϕ) according to existence of a block at designated cellular space coordinate *(i,j)*.

In the cellular space, to guarantee physical consistency, we request the following conditions hold.

C1) no-collision condition: the same position cannot be occupied by multiple blocks (physically infeasible)

$for\ \forall (u_i,u_j) \in q \quad (u_i \neq u_j)$

$h_i \neq h_j\ and\ v_i \neq v_j$

C2) stability condition: each block has to be placed on either table or another stable block.

$for\ \forall u=(X,h,v) \in q$

$assign\ (h,v-1) \neq \phi \qquad (v>1)$

C3) goal attainment condition: the given goal to a particular block is attained only when the blocks which are placed under it has already attained their goals.

$for\ \forall w\ (0<w<v)$

$Y=Z$

where

$Y=assign\ (h,w) \qquad ((Y,h,w) \in q)$

$Z=assign\ (h,w) \qquad ((Z,h,w) \in q_G)$

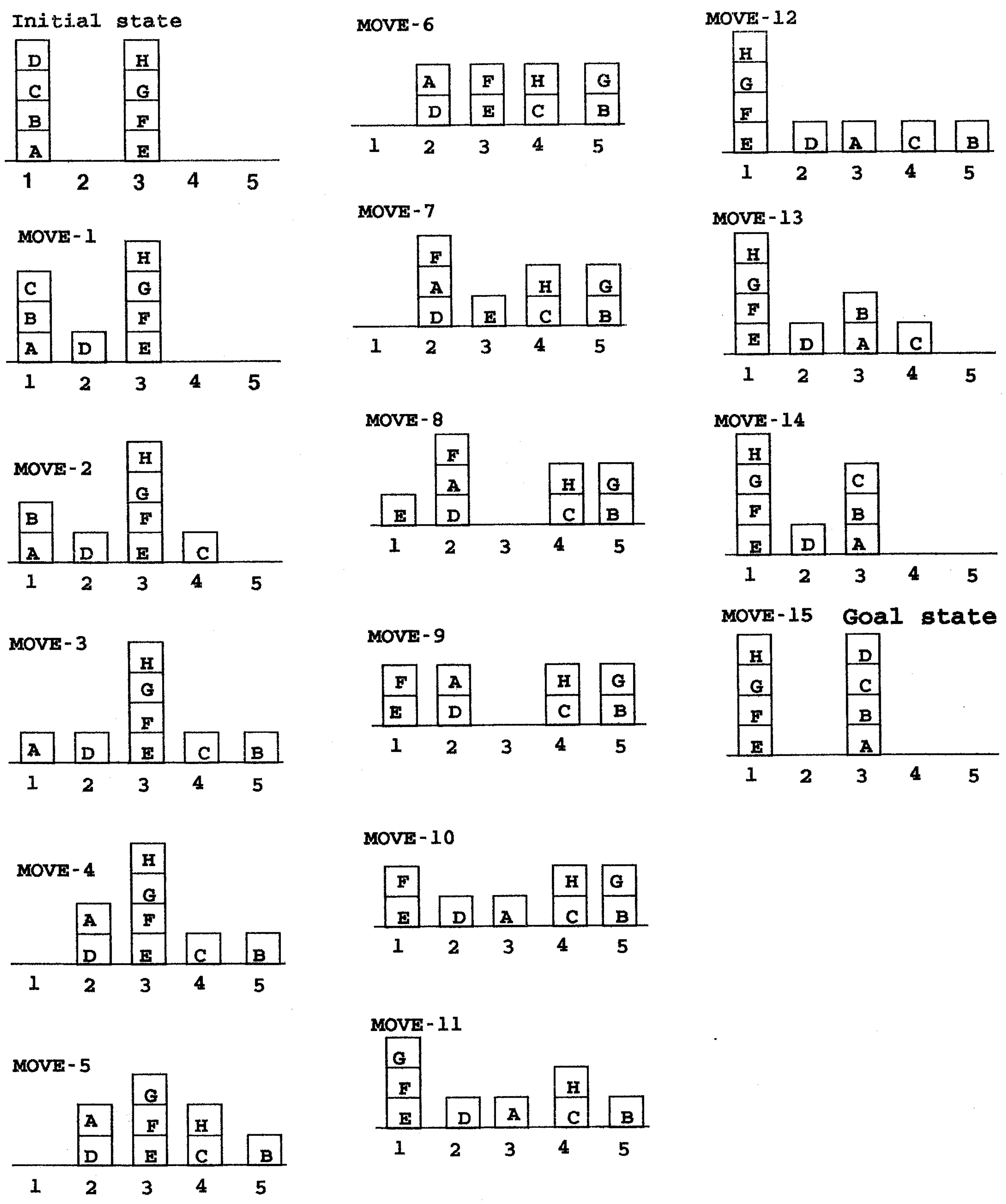

Figure 3 The generated plan for 8-blocks case

4. THE APPROACH METHOD

First, to find the set of input symbol strings S, according to the problem description given above, finite set of input symbols are given as the set of production rules;

$\Sigma=\{r_j \mid j=1,2,\cdots\cdots, N_{BR}\}$

$r_j=(c_j,a_j)$

$c_j=c_1c_2\cdots\cdots c_{NC}$

$a_j=a_1a_2\cdots\cdots a_{NA}$

where c_j is condition part and a_j is the

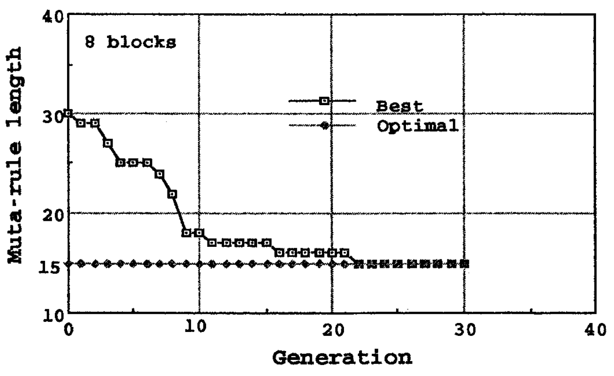

Figure 4 The search result (8 blocks)

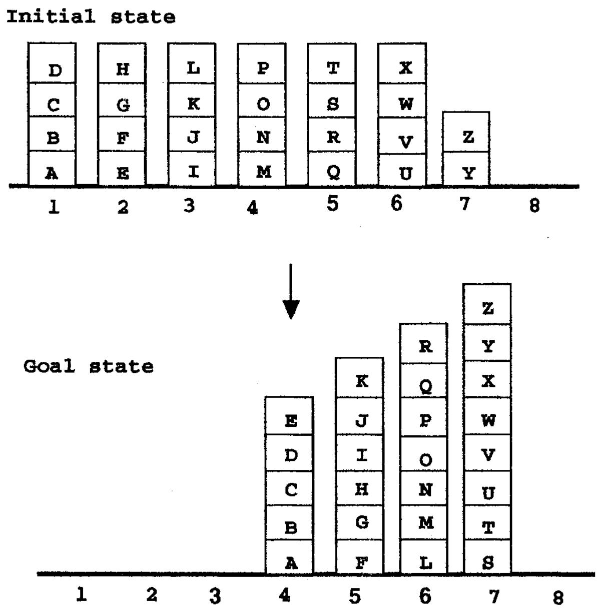

Figure 5 The test example (26 blocks)

action part of the rule. To form a string for GA based search, we define the set of input symbols;

$S=\{R_i \mid i=1,2,\cdots\cdots, N_{PSIZE}\}$

$R_i = mr_1\ mr_2\ mr_{RLEN}$

$(mr_j \in \Sigma)$

The condition part of the BR is given as follows

for $b_i \in B$

$c_1 = GP_i$

$c_2 = WA_i$

$c_3 = 1$: *if* $CG_i = 1$

0: otherwise

$c_4 = 1$: *if* $CG_i = 0$

0: otherwise

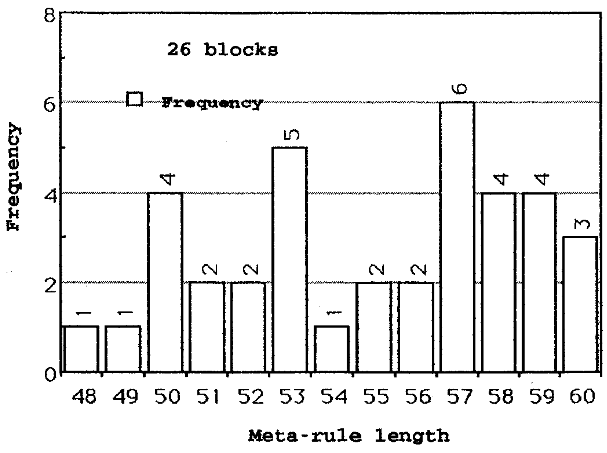

Figure 6 The search performance (meta-rule length) distribution for the 26 blocks test example

where

$GP_i = 1$: *if block i is at a goal position*

0: otherwise

$WA_i = 1$: *if block i is at work area*

0: otherwise

$CG_i = 1$: *if the goal of block i is cleared*

0: otherwise

Finally, we give the action part of BR as

$a_1 = 1$: *move to the goal*

$a_2 = 1$: *move to work area*

This is quite similar to Holland's classifier system[9], but the use of don't care symbols (#) is not considered (i.e. no refinement of rules during the search process).

As mentioned above, the customized GA in this paper is an operator oriented GA. An operator is a gene and, in this problem setting, given as a production rule.

Each string is decoded from left to right position and rules are fired according to current state of the block stack. Given a large number M, the length of MR is given by;

$$RLEN = m: \text{ if } m \leq M, \text{ and } q_m = q_G$$
$$= \infty: \text{ if } m > M$$

where

$$\delta\ (q_0, w_m) = q_m\ (w_m = mr_1 mr_2 \cdots\cdots mr_m)$$

A block to be moved at each step is selected according to the following procedure. The stack of blocks are scanned from left side or right side according to probability calculated by the function *flip-function*. When the j-th BR matches the current state (examined by the function *cond_match*, the block is moved to the designated position by the function *move block*. The function *judge_complete* examines if $q = q_G$.

```
v = False;
j = 1;
while
  j ≤ RLEN or v = False
  sw = flipfunction;
    if sw = 0 then SS = 1: ES = NSTACK
                 else SS = NSTACK: ES = 1;
    for k = SS to ES do
        u = cond_match;
    if u = True then move_block;
  ;
  v = judge_complete;
  if v = True then RSLEN = j;
  j = j + 1;
;
```

NSTACK: the number of horizontal grid positions

Since the population is randomly generated and tested, each MR can have redundant information (i.e. BRs which cannot be fired), we allow genes which are not activated.

$$P_i = p_1 p_2 \cdots\cdots p_j \cdots\cdots p_{PLEN_i}$$
$$(PLEN_i \leq RLEN_i)$$
$$p_j = op: \text{ if } mr_j \text{ fired}$$
$$\phi: \text{ otherwise}$$

After the examination of rule-firing, inactive genes are removed from the corresponding string. Thus we have the variable strings.

Fitness V_i (reward from the problem environment) is given as follows and strings with above-average performance are reproduced (N_i: the expected number of reproduction).

$$N_i = PSIZE \times P_i$$
$$P_i = V_i / \sum_{j=1}^{PSIZE} V_j$$
$$V_i = (RLEN_{max} - RLEN_i)^2$$

5. EXPERIMENTS

In the experiments, we used standard type genetic operations such as crossover, mutation, inversion and deletion. The number of stacked blocks in vertical direction was not limited.

The set of operator symbols Σ and two block stacking examples were given as follows;

$$\Sigma = \{r_1, r_2, r_3, r_4\}$$
$$r_1 = (1000, 10)$$
$$r_2 = (1001, 01)$$
$$r_3 = (0110, 10)$$
$$r_4 = (0101, 01)$$

Example-1 (8 blocks):

q_0 = {(A,1,1), (B,1,2), (C,1,3), (D,1,4), (E,3,1), (F,3,2), (G,3,3), (H,3,4)}

q_G = {(E,1,1), (F,1,2), (G,1,3), (H,1,4), (A,3,1), (B,3,2), (C,3,3), (D,3,4)}

Example-2 (26 blocks):

q_0={(A,1,1), (B,1,2), (C,1,3), (D,1,4),
(E,2,1), (F,2,2), (G,2,3), (H,2,4),
(I,3,1), (J,3,2), (K,3,3), (L,3,4),
(M,4,1), (N,4,2), (O,4,3), (P,4,4),
(Q,5,1), (R,5,2), (S,5,3), (T,5,4),
(U,6,1), (V,6,2), (W,6,3), (X,6,4),
(Y,7,1), (Z,7,2)}

q_G={(A,4,1), (B,4,2), (C,4,3), (D,4,4),
(E,4,5), (F,5,1), (G,5,2), (H,5,3),
(I,5,4), (J,5,5), (K,5,6), (L,6,1),
(M,6,2), (N,6,3), (O,6,4), (P,6,5),
(Q,6,6), (R,6,7), (S,7,1), (T,7,2),
(U,7,3), (V,7,4), (W,7,5), (X,7,6),
(Y,7,7), (Z,7,8)}

Fig.3 shows the generated plan for Example-1. The given goal was attained after 15 steps of block motion. Before testing this example, four small sized examples were tested (4, 6 blocks) and the obtained input strings were inserted before the run. During the search, GA parameters were changed as follows.

Crossover rate	60% to 10%
Mutation rate	0.08% to 0.2%
Inversion rate	0.08% to 0.4%
Deletion rate	0.08% to 0.4%

The parameters were switched when no improvement was observed for three generations. The result shows that the minimum length plan was obtained.

Fig.4 shows the search behavior shown during the above search. The horizontal axis represents generation and the vertical axis represent meta rule length. The substrings are inserted at random position in newly generated strings.

Fig.5 shows the Example-2 (26 blocks included). As well as the first example, this is a nonlinear planning problem. Fig.6 shows the distribution of rule length obtained through the second experiment. The horizontal axis represents the MR length and the vertical axis represents the observed frequency of the MR length. Three smaller size examples (12, 18 blocks) were solves before the run to obtain effective sub-string.

6. CONCLUSIONS

(1) To solve the nonlinear block stacking problem as an optimization problem, I proposed an approach method based on the Operator Oriented Genetic Algorithm. Each gene was coded as an operator rather than pure numerical values.

(2) The block states were represented in cellular coordinate space rather than in propositional form. This had an effect of decreasing the size of search space.

(3) Operators to generate block motions were given as string-formed production rules called basic-rules. A meta-rule was formed as a sequence of the basic-rules. Each meta-rule was coded as an artificial chromosome for GA based search and its capability of goal attainment was tested.

(4) To search plans with minimum motion steps, fitness value was given to each meta-rule and calculated based on its rule length. The length was given according to position of artificial chromosome at which goal state was found.

(5) Based on the proposed methodology,

starting from smaller size problems and accumulating successful substrings, I showed experimental results (for 8, 26 blocks) and the applicability of the methodology.

(6) My future work includes development of hybrid methodology for solving the many-to-many mapping problem by adding some machine learning mechanisms.

REFERENCES

(1) Bylander, T.: Complexity Results for Planning, Proceedings of the 12th IJCAI, pp.274-279, Morgan Kaufmann (1991).

(2) Chenoweth, S.V.: On the NP Hardness of Blocks World, Proceedings of AAAI-91, Vol. 2, pp.623-628, AAAI Press (1991)

(3) Davidor, Y.: Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization. p.164, World Scientific (1991).

(4) Goldberg, D.E.: Genetic Algorithms in Search, Optimization & Machine Learning. p.412, Addison-Wesley (1989).

(5) Erol, K. et al.: On the Complexity of Domain-Independent Planning. Proceedings of AAAI-92, pp.381-386, AAAI Press/ MIT Press (1992)..

(6) Goldberg, D.E. et al.: Don't Worry, Be Messy, Proceedings of ICGA-4, pp.24-30, Morgan Kaufmann (1991)

(7) Gupta, N. and Nau, D.S.: Complexity Results on Blocks-World Planning, Proceedings of AAAI-91, Vol.2, pp.629-633, AAAI Press (1991).

(8) Hirano, H.: Solving the Block Stacking Problem by Genetic Algorithms, Interface, No.2, pp.108-123 (1992)

(9) Holland, J.H. et al.: Induction: Process of Inference, Learning, and Discovery. p.398 (pp.1-150), MIT Press (1986).

(10) Koza, J.R.: Genetic Programming, p.819 (pp.459-469), MIT Press (1992).

(11) Minagawa, M and Kakazu Y.: A Genetic Approach to the Robot Planning Problem: A Study on the Block Stacking Problem, Proceedings of Robotics & Mechatronics Conference, pp.135-138, JSME (1991).

(12) Minagawa, M. and Kakazu, Y.: A Study on Robot Task Plan Generation: A heuristic Approach by GA, Proceedings of 69th JSME Fall Annual Conference, pp.594-596, JSME (1991).

(13) Nilsson, N.J.: Principles of Artificial Intelligence, p.476, Tioga (1980).

(14) Sacerdoti, E.D.: A structure for Plans and Behavior, p.126, Elsevier (1977).

(15) Sacerdoti, E.D.: The Nonlinear Nature of Plans, Readings in Planning, pp.162-170, Morgan Kaufmann (1990).

(16) Suh, J.Y.: Incorporating Heuristic Information Into Genetic Search, Proceedings of the 2nd ICGA, pp.100-107, LEA (1987).

(17) Suh, J.Y.: Operator-Oriented Genetic Algorithm and Its Application to Sliding Block Puzzle Problem, Parallel Problem Solving From Nature 1, pp.98-103, Springer (1990).

(18) Tate, A. et al.: A Review of AI Planning Techniques, Readings in Planning, pp.26-49, Morgan Kaufmann (1990).

(19) Whitehead, S.D. and Ballard, D.H.: Active Perception and Reinforcement Learning, Proceedings of the Seventh International Conference on Machine Learning, pp.179-188, Morgan Kaufmann (1990).