Building-Block Identification by Simultaneity Matrix

Chatchawit Aporntewan and Prabhas Chongstitvatana

Intelligent System Laboratory, Department of Computer Engineering
Chulalongkorn University, Bangkok, 10330, Thailand
43718043@student.chula.ac.th, Prabhas.C@chula.ac.th

We propose a BB identification by simultaneity matrix (BISM) algorithm. The input is a set of ℓ-bit solutions denoted by S. The number of solutions is denoted by $n = |S|$. The output is a partition of bit positions $\{0, \ldots, \ell - 1\}$. The BISM is composed of Simultaneity-Matrix-Construction and Fine-Valid-Partition algorithms. Algorithm SMC is outlined as follows (a_{ij} denotes the matrix element at row i and column j, $\text{Count}_S^{ab}(i,j) = |\{x \in \{0, \ldots, n-1\} : s_x[i] = a \text{ and } s_x[j] = b\}|$ for all $(i,j) \in \{0, \ldots, \ell-1\}^2$, $(a,b) \in \{0,1\}^2$, $s_x[i]$ denotes the i^{th} bit of x^{th} solution, Random(0,1) gives a real random number between 0 and 1).

Algorithm SMC(S)
1. $a_{ij} \leftarrow 0$ for all $(i,j) \in \{0, \ldots, \ell-1\}^2$; // initialize the matrix
   for $(i,j) \in \{0, \ldots, \ell-1\}^2,\ i < j$ **do**
   $a_{ij} \leftarrow \text{Count}_S^{00}(i,j) \cdot \text{Count}_S^{11}(i,j) + \text{Count}_S^{01}(i,j) \cdot \text{Count}_S^{10}(i,j)$;
2. **for** $(i,j) \in \{0, \ldots, \ell-1\}^2,\ i < j$ **do**
   $a_{ij} \leftarrow a_{ij}+$ Random(0,1); // perturb the matrix
3. **for** $(i,j) \in \{0, \ldots, \ell-1\}^2,\ i < j$ **do**
   $a_{ji} \leftarrow a_{ij}$; // copy upper-triangle matrix to lower-triangle matrix
4. return $A = (a_{ij})$; // return the matrix

Algorithm FVP is outlined in the next page. It searches for a *valid* partition for the $\ell \times \ell$ simultaneity matrix M. The main idea is to recursively break the largest valid partition subset $\{0, \ldots, \ell - 1\}$ into many smaller valid partition subsets. If the size of partition subset is smaller than a constant c, the recursion will be terminated.

In the experiment, a simple GA is tailored as follows. Every generation, the SM is constructed. The FVP algorithm is executed to find a valid partition. The solutions are reproduced by a restricted crossover: bits governed by the same partition subset must be passed together. Two parents are chosen by roulette-wheel method. The mutation is turned off. The diversity is maintained.

A single run on a sum of five 3-bit trap functions is shown in the next page. The population size is set at 50. The bit positions are randomly shuffled. It is seen that the partition is improved along with the evolving population. The correct partition is found in the sixth generation. The optimal solution is found in the next generation. The partitioning approach is limited to separable functions. We are looking for a more general approach which will enable the SM to nonseparable functions.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1566–1567, 2003.

Algorithm FVP(M)
return FVP-CORE($\{0, \dots, \ell - 1\}, M$)

Algorithm FVP-CORE(B, M)

```
1. if (|B| ≤ c) then return {B};
2. array L ← {members of B sorted in ascending order};
3. for i = 0 to |B| − 1 do
        array T ← {matrix elements in row i sorted in descending order};
        for j = 0 to |B| − 1 do
            R[i][j] ← x where a_ix = T[j];
        endfor
   endfor
4. P ← ∅;
   for i = 0 to |B| − 1 do
        if (i ∉ B_x for all B_x ∈ P) then
            B_1 ← {i}; B_2 ← {i};
            for j = 0 to |B| − 3 do
                B_1 ← B_1 ∪ {R[i][j]};
                if (VALID(B_1, M) then
                    B_2 ← B_1;
                endif
            endfor
            P ← P ∪ {B_2};
        endif
   endfor
5. P′ ← ∅;
   for B_x ∈ P do
        M′ ← M shrunk by removing row i and column i, i ∉ B_x;
        for b ∈ B_x do replace b with L[b];
        P′ ← P′ ∪ FVP-CORE(B_x, M′);
   endfor
6. return P′;
```

Algorithm VALID(B, M)
return (for all $b \in B$ the largest $|B| - 1$ elements of M's row b
are found in columns of $B \setminus \{b\}$);

Gen.	Partition subsets
1	{0}, {1}, {2}, {3}, {4, 8}, {5}, {6}, {7}, {9}, {10}, {11}, {12, 14}, {13}
2	{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8, 11}, {9}, {10}, {12}, {13, 14}
3	{0, 3}, {1}, {2}, {4}, {5}, {6}, {7}, {8, 11}, {9}, {10}, {12, 14}, {13}
4	{0, 3}, {1, 7, 13}, {2, 5}, {4}, {6}, {8, 11}, {9}, {10, 14}, {12}
5	{0, 3, 9}, {1, 7, 13}, {2, 4, 5}, {6}, {8, 11}, {10, 12, 14}
6	{0, 3, 9}, {1, 7, 13}, {2, 4, 5}, {6, 8, 11}, {10, 12, 14}