Global problem solver for Combinatorial Problems

James Cunha Werner (jamwer2000@hotmail.com)

Tatiana Kalganova (Tatiana.Kalganova@brunel.ac.uk)

Department of Electronic & Computer Engineering, Brunel University

Uxbridge, Middlesex, UB8 3PH

Abstract. Optimisation of combinatorial problems affects the production chain. This happens because resource management, delivery and production plan are essentially considered combinatorial problems. This paper proposes a new approach to code this type of problems by focusing on the scheduling decision point-of-view. The advantages of the proposed approach are (1) the solution obtained by the representation is always feasible, (2) the utilisation of genetic algorithm is easy and (3) the simulation of solution is easy to perform. The generic features of approach are proved by solving different problems such as Travelling Salesman and job shop scheduling.

Keywords: Artificial intelligence, Evolutionary computations, Genetic algorithms, Scheduling, Travelling salesman

1. Introduction

Optimisation is an important concern in modern world. It allows more competitive price and low response time attending the customer without quality loss. Combinatorial Problem class is an important component in the production chain, mainly in manufacturing, because it includes such problems as Travelling Salesman Problem (TSP) and Job Shop Scheduling (JSSP).

To obtain the operational strategy and to maximize the performance, it is necessary design a problem solver which performs well in several different configurations.

There has not been found a global problem solver for this kind of problems [1,2] because of its NP-Hard nature. The difficulty lies in the solution of discrete states and its constraints. Any change in one state value by the optimisation algorithm must be adjusted to comply with the constraints. Usually the optimisation algorithm deals with unfeasible solutions and adjustment routines which spend time and processing resources.

This paper proposes an algorithm to represent solution of combinatorial problems that are always feasible. The paper does not address the limitations of genetic algorithm to deal with local search.

An example of combinatorial problem is the job manufacturing scheduling using 3 machines. The job needs to be processed by machine 1, 2, and 3. If

the optimisation algorithm, in its search for a better solution, changes the last machine to 1, the production plan will be 1, 2, and 1. It is not a feasible solution because machine 3 is not attended and machine 1 is attended twice.

Several techniques were developed earlier for combinatorial problem [17]. They were focused in two points-of-view: permutation representation (PR) and random key representation (RK). PR is used to represent the systems components, for example the jobs and machines. On the other hand, RK tries to order the components using sequences of random numbers.

Usually, the physical components are modelled. This means that decision making process highly depends on the solution configuration. Therefore in our approach the static and dynamic features of system are obtained by modelling the decision process.

In this paper we propose a global problem solver for combinatorial problems. The basic idea of the proposed method, called Decision Modelling Algorithm – DMA, is to change the modelling point of view. Instead of considering the system in terms of components and trying to model it, we propose to model the decision making process. There is a number of candidates in a list each time a decision is made. The decision is the pointer for one candidate of the list. The framework behind this is to divide the complete problem as a sequence of independent decisions. The sequence with best performance index will be the optimal one.

The goal of Combinatorial Problems optimisation is to obtain the best solution with better reliability, adaptability to external and unpredictable events, and fast solutions with less interaction possible. Optimal solutions for combinatorial problems can be obtained with a process simulator, a performance evaluation function and one optimisation algorithm to obtain the best solutions (see Fig. 1). The performance function reflects the goals of the decision maker.

A very important advantage which follows from decision making point-of-view is that only feasible solutions are searched because unfeasible solutions are not included in the candidate list.

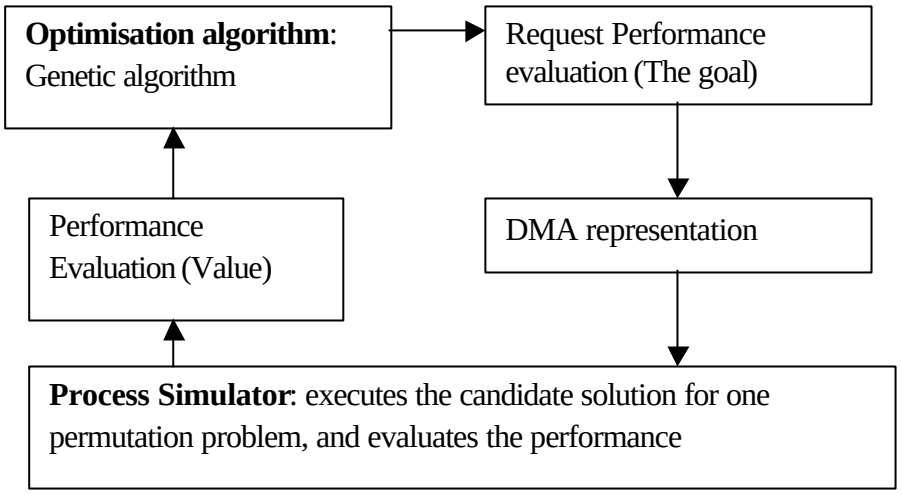

Fig. 1 The proposed DMA architecture for combinatorial problems.

DMA approach is described in detail in section 2. In this paper Genetic Algorithm (GA) is considered as the optimisation algorithm in Section 3. Section 4 describes two production chain applications using the proposed technique: delivery (TSP) and production plan (JSSP). The results of classical techniques are presented and compared with GA approach using DMA. Finally, summary and conclusion are presented in Section 5.

2. The proposed approach: Decision Modelling Algorithm - DMA.

The core of the DMA approach is to obtain the sequence of decisions from a set of values [0,1] that realises one solution of the problem. Let us define the components of DMA approach:

- *Decision factor vector* contains a sequence of real values between 0.0 and 1.0, each value represents one decision.
- *Option vector* contains the allowed candidates for each decision.
- *Option counter (Count_opc)* contains the total number of candidates available in the option vector.
- *Decision sequence vector*: contains the sequence of decisions for the problem.

Fig. 2 illustrates the main principle of the algorithm for each separate decision.

The dynamics of the algorithms can be described for each decision as following:

1. For each decision factor D_i, obtain the pointer for option vector through the product of the factor by option counter.
2. The pointer applied in the option vector shows the candidate to the first decision.
3. Append the decision sequence vector with the pointed candidate.
4. Remove pointed candidate from option vector, shifting left all posterior candidates.
5. The value of option counter reduces by 1 after calculation of one decision.

In section 4 two applications will be solved using this algorithm. The sequence can be executed for each candidate in the list (for deliver problem see subsection 4.1), or every loop a new candidate list can be assembled (for job shop scheduling problem see subsection 4.2).

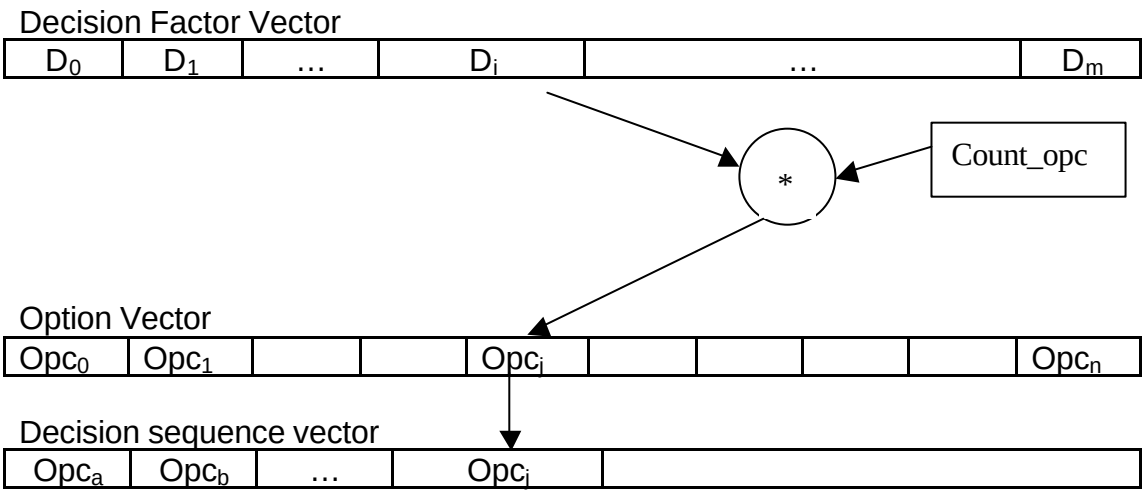

Fig. 2 Decision sequence vector assembly process, where *Count_opc* is the number of candidates in the option vector, *Opc* are the candidate options for each decision, and D_i are the factor values.

3. Genetic algorithm for optimisation.

GA is an optimisation algorithm which mimics the evolution and improvement of life through reproduction. Each individual contributes with its own genetic information to the building of new ones (offspring) adapted to the environment with higher chances of surviving. This is the basis of genetic algorithms and programming [3, 4, 5, 6]. Each 'individual' of a generation

represents a feasible solution to the problem using DMA. In our case the elite simple GA has been used (Fig.3). They code distinct algorithms/parameters to be evaluated by a fitness function.

In our case the following GA operators are used: roulette wheel selection, exchanged mutation, one-point crossover and reinsertion. They perform solution search.

Genetic algorithm software library contains routines code in C-ANSI. The last and actual generation are kept for selection/reinsertion operations. Outputs are written in Excel XLS format direct from the program, to generate an accessible and functional Human-Computer Interface (HCI).

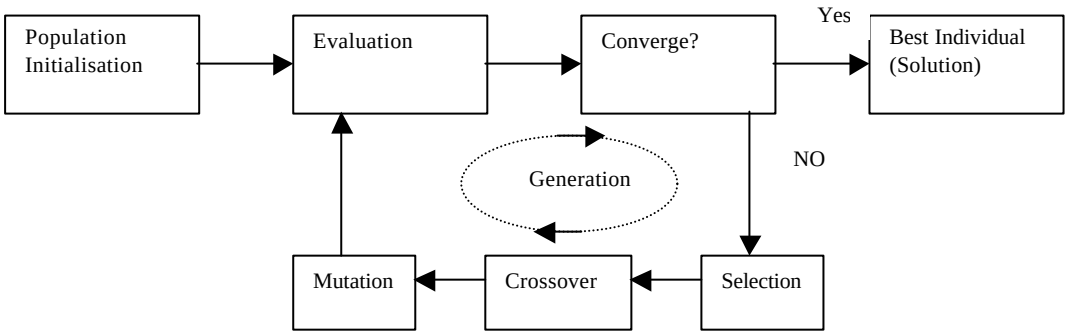

Fig. 3 Genetic algorithm: the sequence of operators and evaluation of each individual.

3.1 Chromosome representation.

Each individual represents a feasible solution. The chromosome of the individual contains the binary representation of the variables that describe one solution of the problem. In our problem the solution can be described as a set of decision values, between 0 and 1. Each value is coded with *n* bits, and refers to the list of possible candidates to be selected. This defines the decision result. The number of bits *n* depends on the possible number of decision candidates:

$$n = \log_2 N^{tot} + 1$$

where N^{tot} is the total number of decision candidates.

Let us consider one example. If there are 9 candidates, it will be necessary 4 bits for each decision, and the chromosome should be represented as shown in Fig.4.

Decision 1	Decision 2	Decision 3	Decision 4

1100	1101	1010	1000

Fig. 4 Binary chromosome representation for a 4 decisions problem.

In this example “Decision 1” is described by binary number I_2=1100 and represents the decimal value I_{10}=1 · 2^3+ 1 · 2^2=12. The number of bits *n* defines into how many values the decision representation interval will be divided. For example, if *n*=4 then there are 16 intervals. The minimum and maximum values of the variable range are defined as [0,1].

The conversion of an integer value *I* to real values *R* follows the relation:

$$\frac{R-0.0}{1.0-0.0}=\frac{I}{2^n} \quad \rightarrow R=\frac{I}{2^n} \tag{1}$$

The pointer for the candidate solution is obtained multiplying the real value *R* by the number of candidates and taking the integer rounded value.

3.2 Fitness function.

The performance of a given solution is defined by the fitness function.

Fitness function uses a definition of performance index to obtain a numeric value that shows how good the solution is. Usually, greater fitness function means better solution.

For each individual it is necessary to evaluate its performance integrating the system models over a time interval. This shows the importance of the simulator in GA optimisation. Using the simulator it is not necessary to actuate in the real system with all individuals, but only with the best one in the generation (Fig. 3).

For each problem, one different performance index is used. For example, delivery system performance is measured by the total distance, and for job shop scheduling is evaluated by the make span time.

The chromosome can be evaluated according to any optimisation equation specified in problem requirements.

3.3 GA implementation.

Genetic algorithm concepts and implementation are summarised in the diagram given in Fig. 5. This diagram shows the operators in action for a population of 4 individuals. The first individual is given in Fig. 4.

Step 0: Population initialization: a random number generator creates random chromosome for each individual.

Population initialization

Step 1 : Parent selection. The best individuals are selected.

	First population	F
	1100110110101000	3.481746
	0101010110110101	3.668023
==>	1000010100110110	6.261380
==>	1101011111001100	12.864222

Step 2: One-point Crossover. Parts of chromosome are exchanged between the 2 chromosomes at a random position.

11010	**11111001100**	------- \		**11010**	10100110110
10000	10100110110	------- /		10000	**11111001100**

Step 3: Mutation. A random bit is exchanged (0 to 1 or 1 to 0).

11**0**10101001**1**0110	1000011111001100
11**1**10101001**0**0110	1000011111001100
F = 8.044	F = 6.092

Step 4: Reinsertion. The offspring replaces the worse individuals.

	second population	F
==>	1111010100100110	8.044
==>	1000011111001100	6.092
	1000010100110110	6.261380
	1101011111001100	12.864222

Step 5: If the result doesn't converge, go to step 1.

Fig. 5 GA operators applied to randomly generated initial population, where *F* is a fitness function (hypothetic value in the example).

4. Experimental results.

In this section we will present two applications solved using the combination of DMA and GA algorithm. They represent different aspects of the proposed approach. One uses the option list to the end, and the other generates a new option list for each decision. The approach can be applied for any other combinatorial problem.

4.1 Application 1. Delivery problems, the Travelling Salesman Problem (TSP).

Delivery problem is an optimisation problem of assigning a route for a vehicle between some nodes.

The algorithm searches for solutions with minimal distance or time between fixed nodes for delivery attending some constraints: each node must

be visited only and at least once. This problem is called *Travelling Salesman Problem (TSP)*.

The most common solution representations used by others authors are described by Chen's [8] detailed review of the state of art: *permutation representation* and *random keys representation*.

Permutation representation (PR) is the most natural representation of a TSP tour. The search space for this representation is the set of permutations of the cities and the chromosome will contain the index of the cities. This representation may lead to illegal tours if the traditional one-point crossover operator is used. Many crossover operators have been investigated for it, with repairing procedures. For example, there are partial mapped crossover (PMX) and order crossover (OX) [8].

Random keys representation (RK) encodes a solution with random numbers from (0,1) [8]. These values are used as sorted keys to decode the solution. For example a chromosome to a 9 city problem may be represented as following:

[0.23 0.82 0.45 0.74 0.87 0.11 0.56 0.69 0.78]

where position *i* in the list represents city *i*. The random number in position *i* determines the visiting order of city *i* in a tour. We sort the random keys in ascending order to get the following tour:

6 – 1 – 3 – 7 – 8 – 4 – 9 - 2 - 5

Random keys eliminate the infeasibility of the offspring by representing solution in a soft manner.

4.1.1 Coding the route using DMA.

DMA decision sequence vector approach is an improvement of RK. It allows the application for problems where the candidates are changed in time, or only different parts of the total net are available in the current moment. The proposed algorithm do not use sort routine that is a bottleneck if a large net is used. Therefore the algorithm can be applied to solution of real-world applications.

The chromosome codes the decisions of route nodes sequence as a string of elements varying between [0,1]. An example of decision sequence vector for delivery problem is shown in Fig. 6. The decision factor vector is the

chromosome of one individual and it contains the solution of the problem. The first value (0.47) of the decision factor vector is multiplied by the number of elements (0.47*10=4.7) in option vector (10) and rounded to point the fifth element (4) of the option vector. This will be the first element of the decision sequence vector. The first visited city will be number 4. City number 4 is removed from the option vector and the counter is updated to 9, because now there are only 9 elements in the option vector.

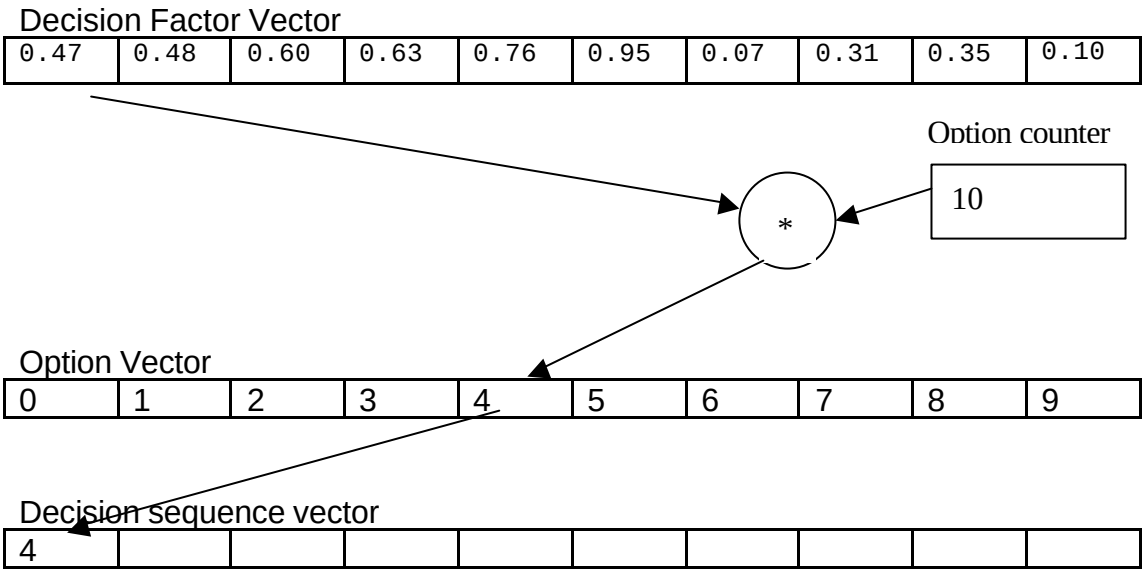

Fig. 6 Decision sequence vector assembly process for the first decision for 9 cities delivery.

The second decision is shown in Fig. 7. The second element of decision factor (0.48) is multiplied by the option counter (9) and results the pointer to option vector (0.48*9=4.3). The element (value 5) is copied to the decision sequence vector, and is removed from the option vector. The option counter is reduced to 8.

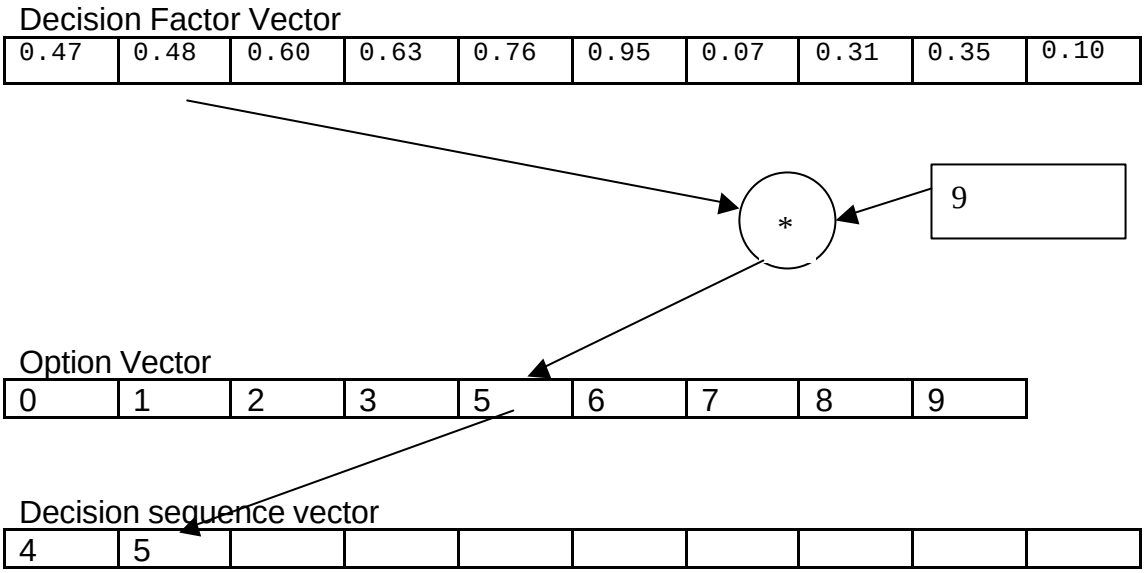

Fig. 7 Decision sequence vector assembly process for the second decision.

The two steps discussed above and shown in Fig. 6 and Fig. 7 are repeated until the end of elements in the option list is reached. After the process has been completed, the option vector is generated:

4	5	6	7	8	9	0	1	2	3

This option vector represents the sequence of the towns visited by salesman. Each individual of the population is interpreted in similar way, and its route is evaluated using a tour simulator (Fig. 1) [7]. The fitness function in this application is the minimisation of the total distance.

4.1.2 Experimental comparison.

Let us consider a net with 10 nodes, and the distances as show in Table 1.

Table 1. Distances between two nodes (bold for minimal values). The margins contains the origin and destination cities, and the intersection value contains the distance.

		Destination									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
ORIGIN	**0**	0.0000	**0.6180**	1.1755	1.6180	1.9021	2.0000	1.9021	1.6180	1.1755	0.6180
	1	**0.6180**	0.0000	**0.6180**	1.1755	1.6180	1.9021	2.0000	1.9021	1.6180	1.1755
	2	1.1755	**0.6180**	0.0000	**0.6180**	1.1755	1.6180	1.9021	2.0000	1.9021	1.6180
	3	1.6180	1.1755	**0.6180**	0.0000	**0.6180**	1.1755	1.6180	1.9021	2.0000	1.9021
	4	1.9021	1.6180	1.1755	**0.6180**	0.0000	**0.6180**	1.1755	1.6180	1.9021	2.0000
	5	2.0000	1.9021	1.6180	1.1755	**0.6180**	0.0000	**0.6180**	1.1755	1.6180	1.9021
	6	1.9021	2.0000	1.9021	1.6180	1.1755	**0.6180**	0.0000	**0.6180**	1.1755	1.6180
	7	1.6180	1.9021	2.0000	1.9021	1.6180	1.1755	**0.6180**	0.0000	**0.6180**	1.1755
	8	1.1755	1.6180	1.9021	2.0000	1.9021	1.6180	1.1755	**0.6180**	0.0000	**0.6180**
	9	**0.6180**	1.1755	1.6180	1.9021	2.0000	1.9021	1.6180	1.1755	**0.6180**	0.0000

For comparison with the proposed method, we select Boltzmann machine (simulate annealing) method source code of which is available in Internet [7] to obtain the optimal solution. We select this approach because it's a local search method with very goods results in the solution of combinatorial optimisation problems. The Boltzmann machine (BM) is a stochastic version of the Hopfield model, whose network dynamics incorporate a random component in correspondence with a given finite temperature. Starting with a high temperature and gradually cooling down, allows the network to reach equilibrium at any step. There are good chances that the network will settle in a global minimum of the corresponding energy function. The network is then used to solve a well-known optimisation problem: The weight matrix is chosen such that the global minimum of the energy function corresponds to a solution of a particular instance of the travelling salesman problem [7].

The experimental results for both approaches are summarised in Table 2. GA parameters are 100 individuals, 60% crossover probability, 5% mutation probability, and the optimal result is obtained after 16 generations.

Table 2. Comparative results of the 9 cities delivery using BM and GA+DMA.

	Distance	**Route**										**Time(s)**
Boltzmann	7.29	8	9	0	1	2	3	5	4	6	7	120
Proposed Method (GA + DMA)	6.18	4	5	6	7	8	9	0	1	2	3	<1

Boltzmann Machine didn't find the optimal value and its processing time is greater than GA+DMA. This result shows that the proposed algorithm is able to obtain the optimal solution coded as a sequence of decisions using DMA and decision represented as a [0,1] value obtained from the chromosome.

4.2 Application 2. Job Shop Scheduling Problem (JSSP).

In this section we will consider second application to our approach – Job Shop Scheduling Problem. In this case, the option vector changes at each decision.

The goal of production scheduling is the optimal allocation of resources over time to perform a set of tasks to attend the customers need. There are

constraints that must be attended, such as sequence, number of machines available, processing time, etc.

Scheduling, especially job shop scheduling, has been studying for a long time [9] – [13]. Some good results have been achieved using Giffler and Thompson algorithms [14]. The method builds schedule considering one task at a time, optimising through permutations the intermediate solution.

Recently, some meta-heuristics such as Simulated Annealing (SA), Taboo Search (TS), and Genetic Algorithms (GA) have been implemented as pure and hybrids methods, where the hybrid methods are superior over pure ones by finding optimal solution [15] – [21]. Standard Genetic algorithms limitations have the disruptive effect of crossover operator, the precocious convergence to some local minima, and the lack of hill climbing (local search methods). Some authors, in order to overcome this limitation developed hybrid methods associating a search algorithm (such as GA) with a local search algorithm (such as SA). In this case, GA obtains the initial parameters condition and the local search evaluates the solution around the initial condition. A complete overview and state of art is available in Chen [8], showing always the representation of the combination of jobs/machines instead of our approach using the decision point of view.

Let us consider the job shop scheduling problem in detail. The JSSP consists of a number of machines, M, and a number of jobs, J. Each job consists of M tasks, each of fixed duration. Each task must be processed on a single specified machine, and each job visits each machine exactly once. There is a predefined ordering of the tasks within a job. A machine can process only one task at a time. There are no set-up times, no release dates and no due dates. The *make span* (or lead time) is the time from the beginning of the first task to start to the end of the last task to finish. The idle time is the total time the machines were waiting for job. There are others optimisation index such as throughput, work in progress, on-line delivery percentage, utilisation (total time – idle time), etc.

The aim of JSSP is to find start times for each task such that the make span time is minimised and no constraints is violated. As a constraint problem, there are M*J variables, each taking positive integer values. The start time of *t-th* task of the *j-th* job is denoted by x_{jt}, and the duration of this

task is defined by d_{jt}. Each job introduces a set of *precedence* constraints on the tasks within that job: $x_{jt} + d_{jt} \quad x_{j(t+1)}$ for $t = 1$ to $M-1$. Each machine imposes a set of *resource* constraints on the tasks processed by that machine: $x_{jt} + d_{jt} \quad x_{pq}$ *or* $x_{pq} + d_{pq} \quad x_{jt}$.

4.2.1 Coding the working plan with DMA.

DMA approach codes a chromosome with M*J values between 0 and 1, one for each decision, which points for the job in the requesting jobs list that win the right to use the machine. Fig. 8 shows an example of chromosome coding based on decision process for the first decision of each machine. The loop carries on until all jobs have been scheduled.

This approach allows us to use the same traditional GA operators without repair the solution to solve the problem because the chromosome contains a sequence of numbers, all numbers represent feasible schedules.

This application differs from delivery because in JSSP for each decision, a new option vector is generated.

4.2.2 Experimental comparison.

Table 3 contains the results for each experiment benchmark using the standard genetic algorithm and DMA. The fitness function minimises the make span time. The total size of the population is 900 individuals. The crossover probability is 80%, mutation probability is 25%, and each decision is coded as 8 bits. The high mutation value is used to keep the population spread over the solution space.

In order to measure our efficiency we compare our results with results from the literature [17], [18], for 3 benchmarks problems [22]: FT06 (6 machines and 6 jobs), FT10 (10 machines and 10 jobs), and FT20 (5 machines and 20 jobs).

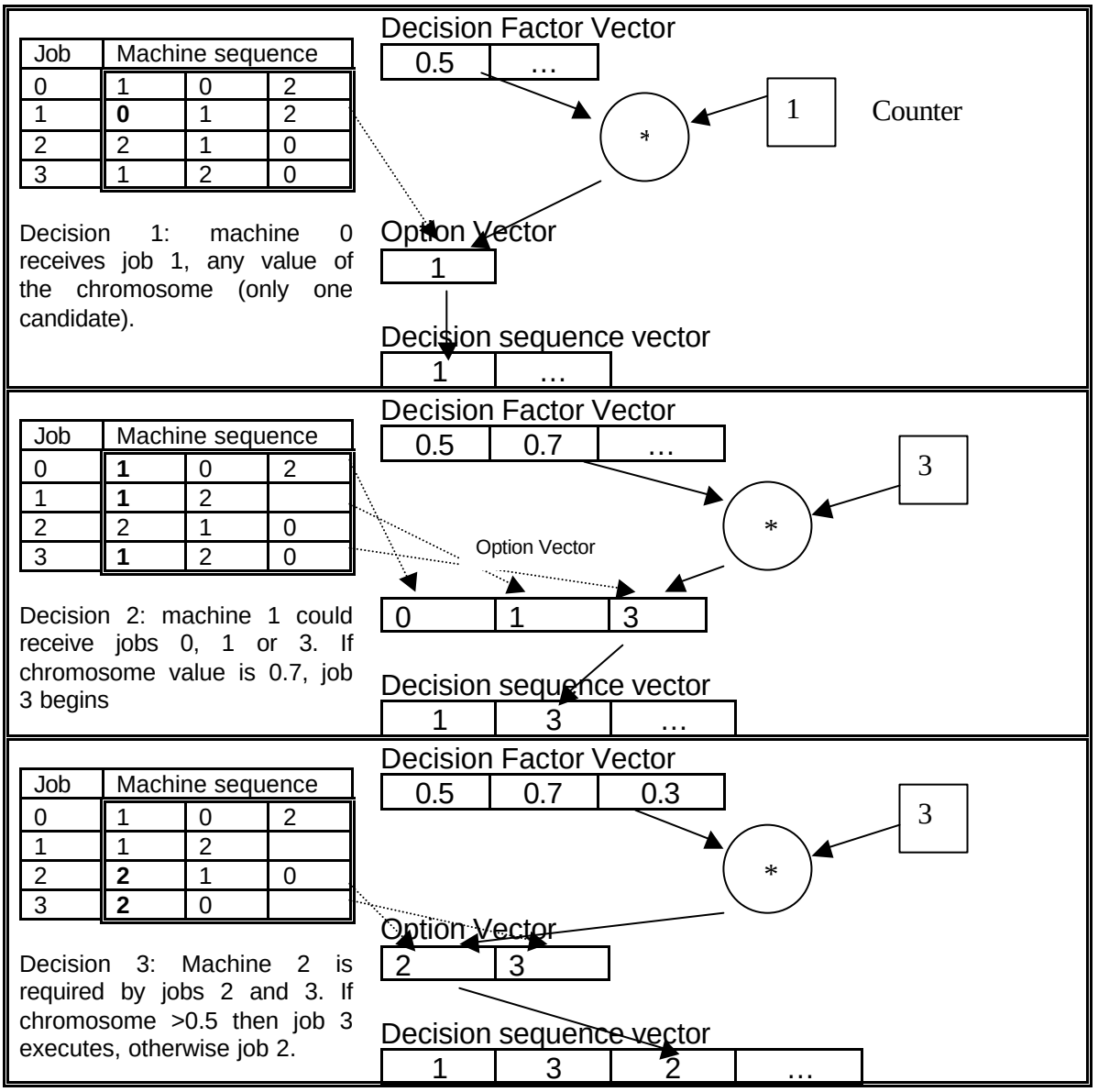

Fig. 8 The three initial decisions sequence in a JSSP example using DMA. In bold are shown the options available for the decision, and will assembly the option vector.

The obtained experimental results are compared with the following algorithms [17]:

1. PRIO: priority rule.
2. RND: priority rules are selected in random.
3. P –GA: priority rule based genetic algorithm.
4. SB1: shifting bottleneck heuristic for 1 machine.
5. SB2: shifting bottleneck heuristic building a partial enumeration tree where each path from root to a leaf is similar to an application of SB1.
6. SB-GA: shifting bottleneck based genetic algorithm.

The results show the proposed algorithm for FT06 is able to obtain the same results as other GA configurations, without any repair or special operators. For FT10 and FT20, the proposed algorithm performs as good as the approaches that do not associate local search algorithms to the genetic algorithm. Unfortunately, there is not available information about the plant floor this benchmark represents to analyse the quality of the solutions. Sometimes, the suboptimal solution is better than the optimal one due stability and reliability to external changes.

Table 3: Comparison of obtained algorithm performance different GA configuration [17] for the benchmark problems FT06, FT10, and FT20 [22].

Method	FT06	FT20	FT10
PRIO	58	1489	1191
RND	58	1374	1088
P-GA	55	1249	960
SB1	57	1274	1031
SB2	55	1240	951
SB-GA	55	1178	938
Proposed Method (GA+DMA)	**55**	**1256**	**1029**

5. Summary and Conclusion

The combinatorial problem class is analysed and a new algorithm based on decision making is proposed. The approach was applied to Travelling Salesman and job shop scheduling problems with competitive results.

The algorithm allows an easy codification of the problem solution configuration and short processing time. The option list can be generated for each decision or not.

The algorithm permits the use of genetic algorithm with conventional operators and chromosome coding real values only without infeasible configurations. The problem of local search using genetic algorithms is not addressed in this paper.

References.

[1] M.R.Garey, D.S.Johnson, 1978, Strong NP-completeness results: motivation, examples and implications, Journal of the ACM, 25:499-508.

[2] M.R.Garey, D.S.Johnson, R.Sethi, 1976, The complexity of flowshop and jobshop scheduling, Mathematics of operations research 1(2):117-129.

[3] L.CHAMBERS, 2000, The practical handbook of Genetic Algorithms, Chapman & Hall/CRC.

[4] J.H.HOLLAND, 1992, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence, Cambridge: Cambridge press.

[5] D.E.GOLDBERG, 1989, Genetic Algorithms in Search, Optimisation, and Machine Learning, Reading, Mass.: Addison-Whesley.

[6] J.R.KOZA, 1992, Genetic programming: On the programming of computers by means of natural selection, Cambridge,Mass.: MIT Press.

[7] K.Kutza; *Neural Networks at your Fingertips*; http://www.geocities.com/CapeCanaveral/1624/

[8] M.Gen, R.Cheng, 1997, Genetic algorithms & engineering design, John Wiley & Sons.

[9] J.Blazewicz, K.H.Ecker, G.Schmidt, J.Weglarz, 1994, Scheduling in computer and manufacturing systems, Springer.

[10] P.Brucker, 1998, Scheduling algorithms, Springer.

[11] S.French, 1982, Sequencing and scheduling, Mathematics and its applications, Elles Horewood Limited.

[12] K. R.BAKER, 1974, Introduction to Sequencing and Scheduling, John Wiley & Son.

[13] E.BALAS, 1969, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, Operations Research, 17, 941-957.

[14] B.Giffler, G.L.Thompson, 1969, Algorithms for solving production scheduling problems, Operations research 8, 487-503.

[15] H.T.Jensen, 2001, Robust and flexible scheduling with evolutionary computation, PhD dissertation, University of Aarhus, Denmark.

[16] P.Ross, 2002, Evolutionary scheduling and routing, Tutorial Gecco2002.

[17] U.Dorndorf, E.Pesch,1995, Evolution based learning in a job shop scheduling environment, Computers Ops. Res. Vol. 22, N.1, pp.25-40.

[18] P.W.Poon, J.N.Carter, 1995, Genetic algorithm crossover operators for ordering applications, Computers Ops. Res. Vol.22, N.1, pp.135-147.

[19] A.S.Jain, S.Meeran, 1999, Deterministic job shop scheduling: past, present and future, European journal of operation research, 113(2) 390-434.

[20] M. E.AYDIN, T. C.FOGARTY, 2002, Simulated annealing with evolutionary processes in job shop scheduling, in Giannakoglou, K., Tsahalis, D., Periaux, J., Papailiou, K. and Fogarty, T. C. (eds.) Evolutionary Methods for Design, Optimisation and Control, (Proc. of EUROGEN 2001, Athens, 19-21 September) CIMNE, Barcelona.

[21] M.VAZQUEZ, D.WHITLEY, 2000, A comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem, 6th Int. Conf. Parallel Problem Solving from Nature – PPSN VI, in Lectures notes in Computing Science vol. 1917, pg. 303-312.

[22] Imperial College Management School: OR Library http://www.ms.ic.ac.uk/info.html