USING THE POPULATION-BASED INCREMENTAL LEARNING ALGORITHM WITH COMPUTER SIMULATION: SOME APPLICATIONS

J. Bekker and **Y. Olivier**[1]

Department of Industrial Engineering
University of Stellenbosch, South Africa
jb2@sun.ac.za

ABSTRACT

The integration of the population-based incremental learning (PBIL) algorithm with computer simulation shows how this particular combination can be applied to find good solutions to combinatorial optimisation problems. Two illustrative examples are used: the classical inventory problem of finding a reorder point and reorder quantity that minimises costs while achieving a required service level (a stochastic problem); and the signal timing of a complex traffic intersection. Any traffic control system must be designed to minimise the duration of interruptions at intersections while maximising traffic throughput. The duration of the phases of traffic lights is of primary importance in this regard.

OPSOMMING

Die integrasie van die *population-based incremental learning* (PBIL) algoritme met rekenaarsimulasie word bespreek, en daar word getoon hoe hierdie spesifieke kombinasie aangewend kan word om goeie oplossings vir kombinatoriese optimeringsprobleme te vind. Twee voorbeelde dien as illustrasie: die klassieke voorraadprobleem waarin 'n herbestelvlak en herbestelhoeveelheid bepaal moet word om koste te minimeer maar nogtans 'n vasgestelde diensvlak te handhaaf ('n stochastiese probleem); en die bepaling van die seintye van 'n komplekse verkeerskruising. Enige verkeerbeheerstelsel moet ontwerp word om die duur van die vloeionderbrekings by verkeerskruisings te minimeer en verkeerdeurset te maksimeer. Die tydsduur van die fases van verkeersligte is dus baie belangrik.

---

[1]The author was enrolled for a B Eng degree in the Department of Industrial Engineering, Stellenbosch University

1. INTRODUCTION

The population-based incremental learning (PBIL) algorithm belongs to the family of metaheuristics used for finding near-optimal solutions to complex problems that are usually at least *NP*-hard. This generally implies problems with large, discrete solution spaces, for which the time to solve the problem grows polynomially with the problem size (number of variables). A classic example of this type of problem is the 'travelling salesperson problem' (TSP, see Winston [23], 519 and 526). Metaheuristics search these large (often infinite) solution spaces in a structured, incremental manner and attempt to find 'good' solutions to problems for which the solutions cannot be found using exact methods or exhaustive enumeration.

There are many metaheuristics available for optimisation. One of the best-known ones is the genetic algorithm (GA) – see for example Goldberg [12], Thomas *et al.* [20], Dereli *et al.* [9], Zhou *et al.* [26], Coit *et al.* [8], and Kreng *et al.* [15] – but simulated annealing, ant colony optimisation, and tabu search are also widely known and applied. The PBIL is perhaps less known, and it is our objective to introduce this algorithm to the local industrial engineering community. Since metaheuristics are often used in combination with computer simulation for optimisation (Fu *et al.*[10]), we present the PBIL algorithm while using computer simulation as the evaluation function.

The PBIL algorithm will now be outlined, followed by discussions of two applications of the PBIL using computer simulation. Aspects of the simulation model and optimisation with the PBIL algorithm are considered, and lastly, the results are presented and evaluated.

2. THE POPULATION-BASED INCREMENTAL LEARNING (PBIL) ALGORITHM

Baluja [3] developed the PBIL algorithm. The basic concepts of the PBIL algorithm are as follows:

Set-up of the solution structure. The solution structure on which the PBIL algorithm operates is similar to that of the general genetic algorithm. The decision variables of the problem are identified and encoded as adjacent binary sub-strings to form a complete string, called a *solution vector*. The nature of the decision variables determines the length of the string. When a decision variable can assume values between, for example, 0 and 100, a sub-string of seven binary digits is sufficient. The digits in the string (solution vector) are randomly assigned binary values (0 or 1). After the sub-strings have been decoded and the appropriate values assigned to the respective variables, such a string constitutes one possible solution. A finite set of these solution vectors forms a *population* of possible solutions; the analyst must select this population size.

Evaluation function. The quality of each possible solution suggested by a metaheuristic must be evaluated – hence the need for an evaluation function. The result of the evaluation function must usually be minimised or maximised. This function is usually a mathematical function (Thomas *et al.* [20], Zhai *et al.* [25]), but for dynamic, complex stochastic systems, simulation models can be used as evaluation functions. This principle is applied in this paper and is shown in Figure 1.
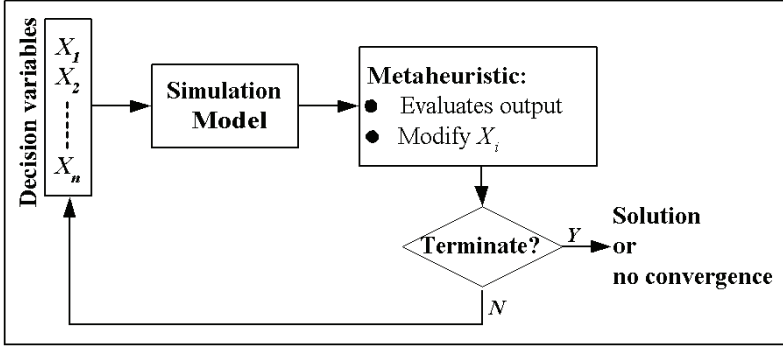
Figure 1: Principle of simulation optimisation with metaheuristics

A large variety of literature exists on simulation optimisation with metaheuristics – see for example Andradóttir [2], Glover *et al.* [11], Baesler and Sepúlveda [5], Law and Kelton [17], Lacksonen [16], Ólafsson and Kim [18], and Truong and Azadivar [21].

Probability vector. The probability vector is an additional structure and has as many elements as the solution vectors, but each element contains a probability value instead of a binary number. The value in a specific element shows the probability that a given digit in a solution vector contains a '1'. A low value in element *i* thus indicates a low probability of finding a '1' in digit *i* of a solution vector. The probability vector is the core of the PBIL algorithm. The elements of the vector are initialised to contain probabilities of 0.5 each, but a selected solution vector from the population modifies the contents of each element during a given iteration.

To illustrate: Suppose the objective function of a given complex problem must be maximised. All the solution vectors in the population are decoded and evaluated through the appropriate evaluation function. The solution vector resulting in the maximum value of the evaluation function is selected from the population to modify the probability vector. This modification is done as follows:

$$P^j(i) \leftarrow P^j(i) \times (1 - LR) + SV_B^j(i) \times LR \qquad (1)$$

where
$P^j(i)$ =   value of the *i*-th cell in the probability vector in the *j*-th generation
LR =   learning rate (typically 0.1–0.4)

$SV_B^j(i)$ =   value of the *i*-th digit in the solution vector (0 or 1) yielding the q
                maximum Evaluation value (the current 'best'), *j*-th generation

A simple example (one variable of four bits and a fictitious evaluation function) to illustrate the above three concepts is shown in Figure 2.
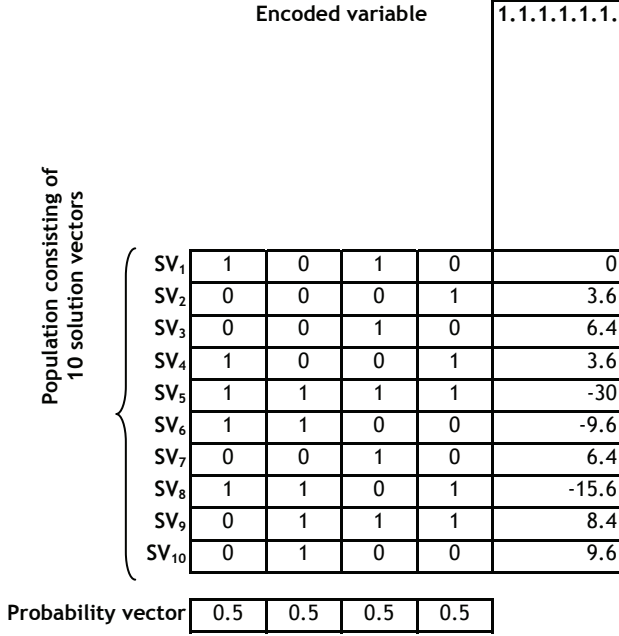
	Encoded variable				1.1.1.1.1.1.
SV₁	1	0	1	0	0
SV₂	0	0	0	1	3.6
SV₃	0	0	1	0	6.4
SV₄	1	0	0	1	3.6
SV₅	1	1	1	1	-30
SV₆	1	1	0	0	-9.6
SV₇	0	0	1	0	6.4
SV₈	1	1	0	1	-15.6
SV₉	0	1	1	1	8.4
SV₁₀	0	1	0	0	9.6

Population consisting of 10 solution vectors (left label)

Probability vector | 0.5 | 0.5 | 0.5 | 0.5

Figure 2: Example of the set-up of the PBIL structure

Mutation. To avoid premature convergence, mutation is applied to the probability vector as follows:

$$P^j(i) \leftarrow P^j(i) \times (1 - MS) + Random(0 \, or \, 1) \times MS \qquad (2)$$

where
MS = the degree to which mutation is applied; a typical value is 0.05.

Mutation is applied with a set probability, typically 0.02. Note that the term *Random*(0 or 1) means that either a '0' or a '1' is selected with probability 0.5.

Formation of the next generation. The solution vectors of the next generation are formed using the probability vector. A random number is created for each element of each solution vector in the population. If the random number is less than the probability value in the corresponding element of the probability vector, a '1' is assigned to that element of the solution vector, otherwise a '0' is assigned. This is repeated for each solution vector in the population.

Convergence. If the algorithm converges, the elements of the probability algorithm converge to either 0 or 1. Threshold values are set to prevent unnecessary iterations; for example if an element reaches a value below 0.05, it is considered to

have converged to 0, while an element that has a value above 0.95 has converged to 1. If all elements satisfy one of these conditions, the algorithm is stopped. A hard-stopping criterion is also used to prevent endless iterations with optimisation problems that behave badly, e.g. 200 iterations. There is of course a risk that the algorithm can stop prematurely while it is converging.

```
//Start Algorithm
1:    Assign LR, Pr(Mutation), MS,      MaxGenerations,     PopulationSize
2:    Initialise Probability Vector P
3:    k←1
4:    Converged←False
```

5: **While Not** *Converged* AND $k \leq MaxGenerations$

6:     **For** j = 1 **To** *PopulationSize*

7:         Generate solution vectors in population

8:         Evaluate the solution vectors

9:     **Next** *j*

10:     Find the best solution vector

    //Update probability vector *P*:

11:     **For** i = 1 **To** *Number of Elements of P*

12:
$$P^k(i) \leftarrow P^k(i) \times (1 - LR) + SV_B^k(i) \times LR$$

13:     **Next** *i*

    //*Do mutation*

14:     **For** j = 1 **To** *Number of Elements of P*

15:         **If** (Random(0,1) < Pr(*Mutation*)) **Then**

16:
$$P^k(j) \leftarrow P^k(j) \times (1 - MS) + Random(0 \text{ or } 1) \times MS$$

17:         **End If**

18:     **Next** *j*

19:     $k \leftarrow k + 1$

20:     *Converged* ← Call **Function** CheckConvergence

21: **End While**

22:     Return Result

```
//End Algorithm
```

Figure 3: General PBIL algorithm

Evaluation function. Each solution vector is decoded from binary format to decimal format, and the value of the evaluation function is determined for the value(s) represented by the given solution vector. The evaluation function can assume many

forms – for example, a mathematical function for which the optimum cannot be determined analytically, or a simulation model, as is the case in this article.

The general algorithm is shown in Figure 3.

Variations of the PBIL algorithm include a negative learning ability (Baluja [3]) as well as duality (Yang and Yao [24]). The PBIL algorithm has been widely used in many applications; see, for example, Chen and Petroianu [6], Al-Sharhan *et al.* [1] and Gosling *et al.* [13]. Baluja applied it to the *j* x *m* job shop scheduling problem (scheduling *j* jobs on *m* machines, an *NP*-complete problem), a 50-city travelling salesperson problem, as well as the bin-packing problem (Baluja [3]).

Computer simulation and the PBIL algorithm were applied to an inventory problem and a traffic intersection (system) problem. These problems are described in the next section.

3. DETERMINING THE REORDER LEVEL AND REORDER QUANTITY IN AN (*s, S*) INVENTORY PROBLEM

The (*s, S*) inventory problem requires a reorder point *s* and a reorder quantity *S* to be determined for a commodity that has a stochastic demand profile. Cost must be minimised while a desired customer service level must be maintained. This problem is well-known, and is covered in almost every operations management textbook (e.g. Hanna and Newman [14], 595-599).

For this application of the PBIL algorithm, assume a simple inventory process in which a single, discrete commodity is sold to customers who arrive according to a Poisson process, with the rate of arrival λ. Assume the demand of customer *i* is distributed $\lceil 20 \cdot beta(2, 1) \rceil$ and the order lead time is *U(1h, 2h)*. The following notation applies:

I_t = inventory level at time *t* when customer *i* arrives

I_0 = starting inventory (at time 0)

s = reorder point

S = reorder quantity

S_L = service level

S_i = stock-out (i.e. inability to fulfil demand) experienced by customer *i*

D_i = demand of customer *i*

N_C = total number of customers that arrived in period *T*

i = customer number at time *t*

In this problem, the service level is defined as the ratio of units of the commodity supplied to the total demand. It is expressed as follows:

$$S_L = \frac{\displaystyle\sum_{i=1}^{N_C} D_i - \sum_{i=1}^{N_C} |S_i|}{\displaystyle\sum_{i=1}^{N_C} D_i} \cdot 100\% \tag{3}$$

The inability to fulfil demand is the stock-out, determined by

$$S_i = \begin{cases} 0: & I_t \geq D_i \\ I_t - D_i & I_t < D_i \end{cases} \tag{4}$$

or $S_i = \min(0, I_t - D_i)$

The commodity inventory consumption profile is shown in Figure 4.

The following assumptions apply:
The system operates for 8 hours every day. The inventory level at the end of a given day is carried over to the following day.

1. The initial inventory is $I_0 = 100$ units.
2. The customer arrival rate is $\lambda = 2/h$.
3. In this problem, *no backlog* is allowed – if $D_i > I_t$ and $I_t > 0$, the customer takes I_t units and after that I_t becomes 0. If $I_t = 0$ and a customer arrives, I_t remains 0, but the stock-out is adjusted according to Equation (4). When the replenishment quantity arrives, I_t is adjusted according to $I_t \leftarrow I_t + S$. Note that the stock-out quantity shown in Figure 4 is for illustration purposes only; our replenishment starts at $I_t = 0$ if a complete stock-out has occurred.
4. The cost to reorder S items is R100/order, and the overall holding cost per item is R10/item-day.

The desired service level is 95%.

The PBIL parameters were: *LR* = 0.3, mutation probability = 0.08 and mutation shift factor *MS* = 0.05. We did not evaluate the sensitivity of each parameter, but found that a learning rate of less than 0.3 caused very slow convergence.A simulation model of this problem was developed, and *s* and *S* were specified as variables that must be altered by the PBIL algorithm. A response function is needed in the simulation model to quantify the effect of the choices for *s* and *S*. The following response function, measured as a pseudo-cost, was used:

$$f = 1000(S_D - \hat{S}_L)^2 + 10\overline{D} + 100\overline{N} \tag{5}$$

where

$\hat{S}_L$ =   the estimated service level resulting from values chosen for *s* and *S*

S_D = the desired service level
$\overline{D}$ = the estimated average daily inventory level
$\overline{N}$ = the estimated number of orders placed per day.
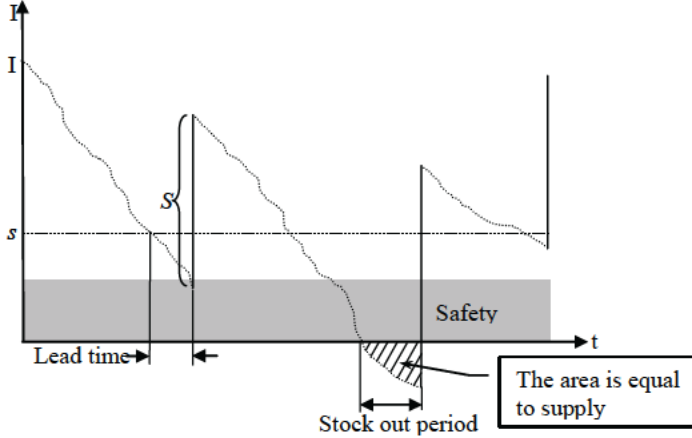

Figure 4: Typical (s, S) inventory profile

In our inventory problem, the value of f must be minimised. The difference $(S_D - \hat{S}_L)^2$ implies that there is always a positive difference between the estimated service level and the desired service level. We chose a quadratic relationship so that the response function could be penalised for both cases of deviation, i.e. when $S_D > \hat{S}_L$ and when $S_D < \hat{S}_L$. The difference is multiplied by a pseudo-cost (i.e. not a tangible cost) of 1,000 to make it dominant for large differences, and comparable to the other terms for small differences. Note that if the difference is 0, the average inventory level and the number of orders placed become dominant. A desired service level can thus not be reached without a minimum inventory level and the minimum number of orders. All these then minimise f.

The estimations were done with the simulation model, using the terminating system approach as prescribed by Law and Kelton ([17] 505-515). Ten independent replications per solution vector of each PBIL generation were executed, for each combination of s and S. Ten replications would generally be insufficient from a statistical perspective, but the simulation model must only provide estimations for the evaluation function; if more replications per combination are executed, the execution time will increase. The analyst thus has to find a trade-off between precision of estimation and computer time. The ranges of s and S were each limited to 2^{10}, or 1,024. The solution space thus contained $1,024^2$ possibilities, and the solution vectors consisted of 20 bits each. The stock-out S_i is often difficult to

determine in practice, but in this problem it was determined by the simulation model, since the demand of each arrival was known.

It was found that s = 97 units, and S = 57 units; the algorithm terminated after 17 generations. The values of s and S were verified by running 100 independent replications of the simulation model. The results are shown in Table 1..

Parameter	Point estimator	Confidence interval half-width (95%)
$\overline{S}_L$	95.05%	0.17%
$\overline{D}$	63 units/day	3.90
$\overline{N}$	3.57 orders/day	0.14

Table 1: Results for estimating the (s, S) parameters with 100 replications

The convergence of the two parameters is shown in Figure 5. The mean values per generation for the variables s and S are shown for each generation.
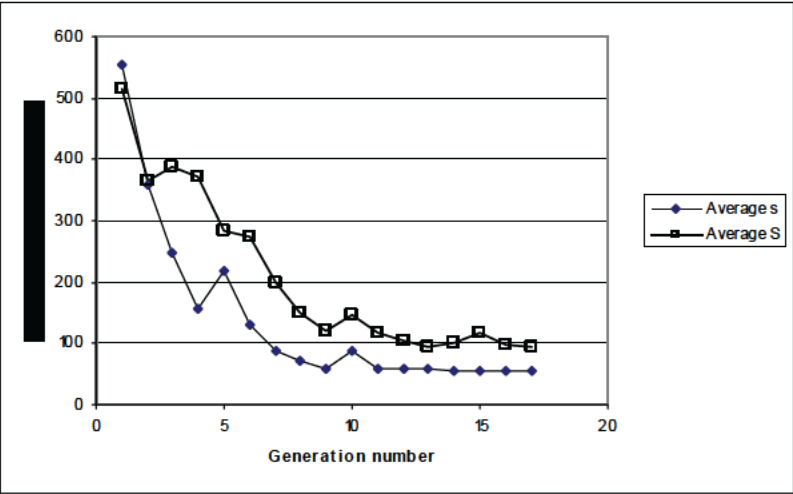

Figure 5: Convergence of the mean response function value for the (s, S) inventory problem

It took less than three minutes for the problem to converge on a Dell computer with two 1.66 GHz processors.

Next, the use of the PBIL algorithm for a more complex, practical optimisation problem will be demonstrated.

4. DETERMINING SIGNAL TIMING AT A COMPLEX TRAFFIC INTERSECTION

Traffic streams require the option of being diverted or converged, which imposes conflict. These conflicts may be reduced by space or time separation of the streams (Chou *et al.* [7]). Time separation requires signal control, which in its simplest form provides a sequence of green, orange (amber) and red displays. The duration of these display phases is critical for optimum control of traffic, and must be well planned to minimise vehicle delay and simultaneously maximise vehicle throughput at the control point.
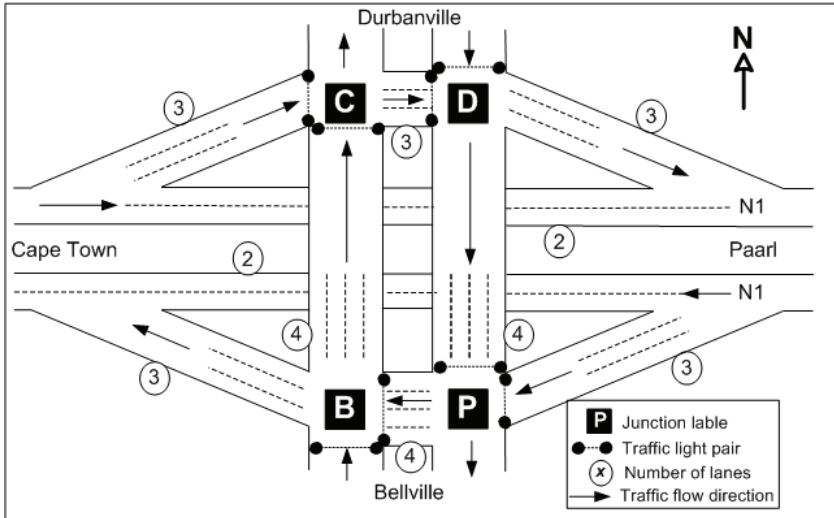

Figure 6: Schematic plan view of the intersection investigated (not to scale)

Computer simulation enables one to study dynamic systems (Law and Kelton [17]), and can therefore also be applied in traffic control system design, as was shown by Chou *et al.* [7]. The effect of changing the durations of phases can be observed for a given arrival pattern of vehicles. A large number of possible values can be selected for the phase durations, which results in a large solution space. If one accepts integer values for the phase durations, combinatorial optimisation with metaheuristics is possible. As explained previously, a metaheuristic linked to simulation enables one to improve dynamic (and often stochastic) systems to near-optimality. In this case, the PBIL algorithm will specifically be used. Although not new [19], the combination of computer simulation and the PBIL is applied to improve the throughput of a traffic light control system. Advanced traffic signal timing and optimisation can be achieved with dedicated applications, e.g. *Synchro* and the various forms of *Passer* II, III and IV [19].

4.1 Traffic intersection system description

A relatively complex traffic intersection system exists where the national road (N1) is connected to the two suburbs of Bellville and Durbanville in the Western Cape, South Africa. The N1 consists of two lanes per direction, while exit and entry routes connect it to streets running across the N1 via an overpass. These streets have three or four lanes in each direction. A schematic plan of the intersection is shown in Figure 6. (Note that vehicles in South Africa keep left, as opposed to keeping right in many other countries.) Circled numbers indicate the number of lanes per route.

At each junction there is a set of traffic lights that operate in pairs. These lights provide the typical display sequence of green, orange/amber and red to control traffic flow. The system is pre-timed, which means that it repeats a preset constant cycle that constitutes the sum of the durations of the three display colours. Chou *et al.* [7] explain the pre-timed principle.

Table 2 shows the results of traffic counts that were made at this intersection during the morning peak hour.

Arriving from (at junction)	Number of vehicles arriving per hour	Depart to	Number of vehicles departing per hour	Percentage
Paarl (P)	1,120	Bellville	520	46
		Durbanville	600	54
Bellville (B)	1,500	Cape Town	420	28
		Durbanville	870	58
		Paarl	210	14
Cape Town (C)	2,330	Durbanville	1,480	64
		Bellville	850	36
Durbanville (D)	2,510	Paarl	380	15
		Bellville	1,000	40
		Cape Town	1,130	45

Table 2: Traffic volumes during peak period (07:00-08:00)

4.2 Implementation

The simulation model of the system described above was developed in Arena 9.0, and the PBIL algorithm was coded in Visual Basic for Applications (VBA). The VBA code executes the algorithm using the Arena model as the evaluation function. This is repeated until convergence takes place or the termination criterion is satisfied.

4.3 Assumptions and simplification

The following assumptions were made:

- When the green phase starts, the first vehicle in each lane that may continue will take longer to proceed through the system than the vehicles following it, because of the starting delay. This time is taken as 5 seconds, while the other vehicles take 3 seconds each to proceed through the system. These times are only estimates, and may be different in other studies. Chou *et al*. [7] took the starting delay as 0 seconds, and consecutive delays of 2.2 seconds per vehicle.

- The duration of the orange/amber phase is a constant. It will be divided equally between the red and the green time and will therefore be ignored.

- Although vehicles arrive according to a varying daily pattern at the various junctions, it was assumed that they arrive at a constant arrival rate during the peak hours of the day.

- No distinction is made between types of vehicles – for example, motorcycles or cars.

- The space in the system is limited (see Figure 6), and it is assumed that on average a vehicle occupies five metres of a lane. When these internal lanes are fully occupied, vehicles wishing to enter the system from other lanes cannot proceed even if they receive the green signal. The distance between the junctions marked 'P' and 'D' in Figure 6 is 130 metres, while the distance between 'P' and 'B' is 80 metres. The system is symmetric.

- The period studied was from 07:00 to 07:30, which is a reasonable representation of the worst conditions during the peak hour. The simulation model was allowed to 'warm up', i.e. to reach a steady state during each replication, after which the system performance was measured for 1,800 seconds. The vehicles in process are disposed at the end of the simulation run.

4.4 Input variables and performance measures

The input variables that are controlled by the PBIL algorithm are the time durations for the phases of the traffic light sets. The variables are only defined for the traffic lights giving way to the traffic arriving in the system, i.e. for traffic arriving from Paarl at junction 'P', from Bellville at junction 'B', from Cape Town at junction 'C' and from Durbanville at junction 'D' (see Figure 6). This approach is followed because the opposite traffic lights in each set are always in the opposite state for the same time duration as the given traffic light set, which means the number of

variables manipulated by the algorithm can be reduced. Each traffic light set also has an *offset time*, which is a once-off time delay starting at $t = 0$. Before the timing of a set starts, it is thus delayed by this period, which determines the synchronisation of the system. The times to be varied are as follows:

- *Paarl_Green* – Duration of the green phase for the traffic light system at the junction receiving traffic from Paarl and Durbanville.

- *Bellville_Green* – Duration of the green phase for the traffic light system at the junction receiving traffic from Bellville and Paarl.

- *CapeTown_Green* – Duration of the green phase for the traffic light system at the junction receiving traffic from Cape Town and Bellville.

- *Durbanville_Green* – Duration of the green phase for the traffic light system at the junction receiving traffic from Durbanville and Cape Town.

- *Paarl_Offset* – Offset time duration from $t = 0$ before the timing cycle starts, for the traffic light system at the junction receiving traffic from Paarl and Durbanville.

- *Bellville_Offset* – Offset time duration from $t = 0$ before the timing cycle starts, for the traffic light system at the junction receiving traffic from Bellville and Paarl.

- *CapeTown_Offset* – Offset time duration from $t = 0$ before the timing cycle starts, for the traffic light system at the junction receiving traffic from Cape Town and Bellville.

- *Durbanville_Offset* – Offset time duration from $t = 0$ before the timing cycle starts, for the traffic light system at the junction receiving traffic from Durbanville and Cape Town.

- *Cycle_Time* – The cycle time is the sum of the three colour phases of a traffic light set, and must be the same for all sets for the system to be synchronised. The duration of the green phase and the red phase need not be equal, however.

The relationships among the traffic light control times for a simple traffic intersection can be schematically explained as shown in Figure 7. Note that the exact starting positions and durations are not important for the purpose of the example.

The duration of the red phase is determined by subtracting the duration of the green phase from the duration of the cycle time. Practical requirements state that the minimum duration of the green phase is seven seconds, while the maximum duration is 127 seconds. The nine variables are therefore encoded using seven-bit binary numbers. The algorithm will only find integer values in one-second resolution for the solutions, although time is a continuous phenomenon.
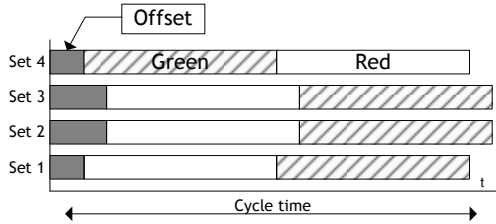
Figure 7: Traffic light control times

The measures of performance are:

- The *total number of vehicles* that left the system over the simulation time period.

- The *mean time in the system*. The time in the system is the period measured from the moment the vehicle enters the system until it leaves. The system could be entered while all lanes at an entry point are open, or when a queue of significant length has developed in a given lane at that entry point. This value is to be minimised.

Although these two parameters are related (i.e., when the number of vehicles per unit time serviced by the system increases, the mean time in the system must decrease), both are measured since they must be quantified, while the first parameter is used for validation.

4.5 Model validation

Simulation is the imitation of a real-world process over time (Banks [4]), and before results can be generated with such a model, it must be validated. Validation is the process of ensuring that the model is a sufficient representation of the real-world process under study, for the particular objectives of the study (Law and Kelton [17]). At least 75 validation techniques exist; in this case the model was subjected to informal techniques, which include *Desk checking* and *Face validation* (Banks [4]). The traffic counts of the real system were compared to those of the simulation model (based on several independent replications), and the results are shown in Table 3 (Face validation). In this validation run, the various phase durations in the model were set equal to those of the real system.

The model was considered to be valid, based on the reasonably small differences in traffic counts.

4.6 PBIL algorithm parameters

The following parameters were used for the application of the PBIL algorithm to this problem:

66

Lane leading to traffic light*	Number of vehicles entering system (Real-world)	Number of vehicles entering system (Simulated)	Difference (%)
D to P	560	564	0.71%
Paarl to P	1,540	1,486	-3.63%
Bellville to B	750	751	0.13%
P to B	865	844	-2.49%
C to Durbanville	1,165	1,202	3.08%
B to C	845	863	2.09%
Durbanville to D	1,255	1,287	2.49%
C to D	540	504	-7.14%
Total number of vehicles entered	3,730	3,797	1.76%

*(see Figure 6)

Table 3: Traffic counts (07:00-07:30) versus simulation results of the system under study

- Learning rate = 0.2
- Mutation shift = 0.05
- Mutation probability = 0.08
- Generations = 50
- Solution vectors in the population = 30

The sensitivity of the values of these parameters was not evaluated, but recommended values from the literature (Baluja [3]) were accepted.

4.7 Results: traffic intersection

Since the duration of the green phase cannot be less than seven seconds, a condition was added that causes the algorithm to assign a very large value to the evaluation function if the duration of any green phase is less than seven seconds. The results are shown in Table 4.

The point estimators and interval estimators (h) (95% level of confidence) for the performance measures based on 50 independent replications of 1,800 seconds each are shown in Table 5.

The paired t-test (Walpole and Myers [22]) is applied to determine if the real-world model and the PBIL-based proposal are significantly different, based on 50 independent replications at the 95% level of confidence. The results are shown in Table 6.

Variable	Values (seconds)	
	Using PBIL	Real values
Offset time Paarl, junction P	96	59
Junction Paarl Green	63	24
Offset time Bellville, junction B	76	51
Junction Bellville Green	63	27
Offset time Cape Town, junction C	57	57
Junction Cape Town Green	114	35
Offset time Durbanville, junction D	12	65
Junction Durbanville Green	89	34
Cycle time	171	75
Average time spent in the system	63.5 seconds	100.7 seconds*

* Obtained from the output of the validation model.

Table 4: Traffic light control variable values suggested by the PBIL algorithm

	Real-world model		Using PBIL	
	Average	h*	Average	H
Time in system (s)	100.7	0.6	63.5	1.3
Vehicles processed	3,291	2	3,750	5

*Confidence interval half-width

Table 5: Point and interval estimators for the performance measures

The expected time a vehicle spends in the real-world system is thus greater than the time it would spend in a system that used the signal times proposed by the PBIL algorithm.

Direction of subtraction	Estimated mean difference (s)	Half-Width
Real-world model – PBIL-based	37.3	5.34
Reject the null hypothesis Ho: Means are equal at 95% level of confidence		

Table 6: Results of the paired *t*-test of the expected time in system

The paired *t*-test is also applied to determine if the number of vehicles processed during the period of observation differs significantly. The results are shown in Table **7**.

Direction of ubtraction	Estimated mean difference (s)	Half-width
Real-world model – PBIL-based	-459	5.16
Reject the null hypothesis Ho: Means are equal at 95% level of confidence		

Table 7: Results of the paired *t*-test of the number of vehicles processed

The expected number of vehicles processed by each system during the period of observation differs significantly – the system using signal times proposed by the PBIL algorithm processes more vehicles than the real-world system.

Although the research is unable to claim that the results are optimal, they do show a significant improvement over the current real-world situation.

All the results are of course only valid for the time period under study, i.e. 07:00-07:30. It took approximately 12 minutes for the problem to converge on a Dell computer with two 1.66GHz processors. The analysis could be repeated for other time periods, provided the arrival rates were known. In doing so, an adaptable traffic control system could be realised, since the signal timings could be adjusted according to the time of day and even the day of the week.

5. CONCLUSION

This article focused on the integration of the PBIL algorithm with computer simulation. It was shown how this combination could be applied to improve real-world systems, in particular the reorder level and reorder quantity of an (s, S) inventory system. The application of the PBIL algorithm to this problem was presented as a starting point, while the phase durations of a traffic light system at a relatively complex traffic intersection were studied as a second, realistic problem. The PBIL algorithm, in conjunction with simulation, showed that the expected time vehicles spend in the system is significantly lower, which implies that a higher traffic volume could be handled. Various other periods of the day should be studied to provide parameters for the system to be adapted to changing traffic situations.

In general, industrial engineers should take note of the PBIL algorithm and its potential in optimisation, especially when used with computer simulation. Also, if traffic authorities could be convinced of the validity of this analysis, it could provide them with a profitable solution to traffic situations of this kind.

6. REFERENCES

[1] **Al-Sharhan, S., Karray, F. and Gueaieb, W.** 2001. *Approach of optimizing computer networks using soft computing techniques.* Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SOFTCOM'01), 847-854.

[2] **Andradóttir, S.** 1998. *A review of simulation optimization techniques.* Proceedings of the 1998 Winter Simulation Conference, The Institute of

Electrical and Electronics Engineers (IEEE), Piscataway, NJ 08855-1331, USA, 151-158.

[3] **Baluja, S.** 1994. *Population based incremental learning: A method for integrating genetic search based function optimisation and competitive learning*. Technical Report, CMU-CS-94-163. Carnegie Mellon University, Pittsburgh, PA 15213, USA.

[4] **Banks, J.** 1998. *Handbook of simulation: Principles, methodology, advances, application, and practice*, John Wiley & Sons, Inc., New York, NY.

[5] **Baesler, F. and Sepúlveda, J.A.** 2000. *Multi-response simulation optimization using stochastic genetic search within a goal programming framework*. Proceedings of the 2000 Winter Simulation Conference, The Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ 08855-1331, USA, 788-794.

[6] **Chen, L. and Petroianu, A.** 1998. *Application of PBIL to the optimization of PSS tuning*. 1998 International Conference on Power System Technology Proceedings, 2, 834-838.

[7] **Chou, C., Chen, C. and Li, M.C.** 2001. Application of computer simulation to the design of a traffic signal timer, *Computers & Industrial Engineering*, 39(1-2), 81-94.

[8] **Coit, D.W. and Smith, A.E.** 2002. Genetic algorithm to maximize a lower-bound for system time-to-failure with uncertain component Weibull parameters, *Computers & Industrial Engineering*, 41(4), 423-440.

[9] **Dereli, T. and Filiz, I.H.** 1999. Optimisation of process planning functions by genetic algorithms, *Computers & Industrial Engineering*, 36(2), 281-308.

[10] **Fu, M.C., Glover, F.W. and April, J.** 2005. *Simulation optimization: A review, new developments, and application*. Proceedings of the 2005 Winter Simulation Conference, The Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ 08855-1331, USA, 83-95.

[11] **Glover, F., Kelly, J.P. and Laguna, M.** 1999. *New advances for wedding optimization and simulation*. Proceedings of the 1999 Winter Simulation Conference, 255-260.

[12] **Goldberg, D.E.** 1989. *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Boston, MA.

[13] **Gosling, T., Jin, N. and Tsang, E.** 2004. *Population based incremental learning versus genetic algorithms: Iterated prisoners dilemma*. Technical Report, CSM-401. University of Essex, Essex, England.

[14] **Hanna, M.D. and Newman, W.R.** 2007. *Integrated operations management*, 2nd edition, Thomson South-Western, Mason, OH.

[15] **Kreng, V.B. and Lee, T.** 2004. Modular product design with grouping genetic algorithm – a case study, *Computers & Industrial Engineering*, 46(3), 443-460.

[16] **Lacksonen, T.** 2001. Empirical comparison of search algorithms for discrete event simulation, *Computers & Industrial Engineering*, 40(1-2), 133-148.

[17] **Law, A.M. and Kelton, W.D.** 2000. *Simulation modeling and analysis*, 3rd edition, McGraw-Hill, Boston, MA.

[18] **Ólafsson, S. and Kim, J.** 2002. *Simulation optimization*. Proceedings of the 2002 Winter Simulation Conference, The Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ 08855-1331, USA, 79-84.

[19] **Sabra, Wang & Associates.** 2003. *Signal timing process final report*. U.S. Department of Transportation http://ops.fhwa.dot.gov/arterial_mgmt/rpt/sig_tim_proc/index.htm, (accessed 13 June 2007)

[20] **Thomas, G.M., Gerth, R., Velasco, T. and Rabelo, L.C.** 1995. Using real-coded genetic algorithms for Weibull parameter estimation, *Computers & Industrial Engineering*, 29(1-4), 377-381.

[21] **Truong, T. and Azadivar, F.** 2003. *Simulation based optimization for supply chain configuration design*. Proceedings of the 2003 Winter Simulation Conference, The Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ 08855-1331, USA, 1268-1275.

[22] **Walpole, R.E. and Myers, R.H.** 1993. *Probability and statistics for engineers and scientists*, 5th edition. Macmillan Publishing Company, New York, NY.

[23] **Winston, W.L.** 1994. *Operations research applications and algorithms*, 3rd edition, Wadsworth, Inc., Belmont, CA.

[24] **Yang, S. and Yao, X.** 2005. Experimental study on population-based incremental learning algorithms for dynamic optimization problems, *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 9(11), 815-834.

[25] **Zhai, L., Khoo, L. and Fok, S.** 2002. Feature extraction using rough set theory and genetic algorithms – an application for the simplification of product quality evaluation, *Computers & Industrial Engineering*, 43(4), 661-676.

[26] **Zhou, H., Feng, Y. and Han, L.** 2001. The hybrid heuristic genetic algorithm for job shop scheduling, *Computers & Industrial Engineering*, 40(3), 191-200.