Solving University Timetabling As a Constraint Satisfaction Problem with Genetic Algorithm

Teddy Wijaya and Ruli Manurung
Faculty of Computer Science, University of Indonesia
Email: tedd50@gmail.com, maruli@cs.ui.ac.id

***Abstract*—The process of manually creating a university timetable is a laborious and error-prone task due to the multitude of constraints that must be satisfied. This paper proposes a method to automate this process. It models the task of producing a course timetable as a multi objective problem that must take into account various resources such as the curriculum, lecturers, classrooms, and time slots. More specifically, in this paper we represent the timetabling problem as a constraint satisfaction problem which we then solve using the Genetic Algorithm (GA). We present our modeling of real-world constraints encountered at the Faculty of Computer Science, University of Indonesia. One of the key issues we explore is the scheme for representing a potential timetable as a chromosome string. Four different schemes are proposed, some of which encode certain heuristics about the domain problem. Our experimental results show that the choice of representational scheme significantly affects the quality of the yielded solution, and that for modestly-sized input problems the GA can successfully solve timetables that satisfy all constraints.**

I. Introduction

COURSE timetabling is a routine activity which must be carried out by every educational institution. A poorly-designed timetable can cause various problems, such as a student not being able to enroll in the course that she wants because the course conflicts with a compulsory course. Consequently, good timetabling is needed to support course activities and provide benefits to students and lecturers.

The process of manually creating a university timetable is a laborious and error-prone task due to the multitude of constraints that must be satisfied. In this paper we propose a method to automate this process. More specifically, we represent the timetabling problem as a constraint satisfaction problem which we then solve using the Genetic Algorithm (GA). In Section 2 we provide background to timetabling, constraint satisfaction problems, and solving multi objective optimization problems using the Genetic Algorithm (GA). In Section 3 we present our modeling of real-world constraints encountered at the Faculty of Computer Science, University of Indonesia, including our various proposals for chromosome representation schemes. Section 4 presents our experimental design and results.

II. Timetabling as CSP

In previous research [2], timetabling is represented as a constraint satisfaction problem (CSP) and solved in two steps using the branch-and-bound mechanism. The first step is teaching assignment, i.e. mapping courses to lecturers, and the second step is timetabling: mapping courses to time slots and classrooms.

In this paper, the timetabling problem is represented as a CSP and then solved using the Genetic Algorithm (GA). The mapping of courses, lecturers, classrooms, and time slots is done in a single step, thus leaving the opportunity as wide open as possible to satisfy the various constraints.

A. Timetabling

University timetabling is a problem in creating a schedule of lectures with limited resources available while satisfying the constraints on the limited resources. The resources in this case are courses, lecturers, classrooms, and time slots. For example, one obvious constraint is that a lecturer cannot teach more than one course at the same time.

B. Timetabling as CSP

A constraint satisfaction problem (CSP) is defined as a set of variables $\{X_1, X_2, \ldots, X_n\}$, and a set of constraints $\{C_1, C_2, \ldots, C_m\}$. Every variable X_i has a non-empty domain D_i. Every constraint C_i involves some subset of variables and determines allowable values from this subset. A problem state is then defined as an assignment to some or all variables, $\{X_i=v_i, X_j=v_j, \ldots\}$. An assignment that does not violate the various constraints is called a consistent or legal assignment. The solution to a CSP is a complete assignment that satisfies all the constraints [1].

C. Solving CSPs with Genetic Algorithms

Finding a solution to a CSP is a non-trivial computational task. One commonly applied solution is to employ the genetic algorithm (GA). The genetic algorithm is a local search algorithm that uses the principle of natural selection and genetic inheritance, where the best individual can survive and produce descendants which inherit characteristics from its parents. Essentially it is a heuristic search strategy that relies on random traversal of a search space with a bias towards more promising solutions. Specifically, it evolves a population of individuals over time, through an iterative process of evaluation, selection, and evolution. Upon termination, the fittest individual is hoped to be an optimal, or near-optimal, solution [4]. The pseudo code of the Genetic Algorithm can be seen in Fig.1.

```
function GENETIC-ALGORITHM(population, FITNESS)
returns an individual
    inputs:   population, a set of individuals
              FITNESS, a function that measures the fitness of
              an individual
    repeat:
        new_population ← empty set
        loop for i from 1 to SIZE(population) do
            x ← RANDOM-SELECTION(population,FITNESS)
            y ← RANDOM-SELECTION(population,FITNESS)
            child ← REPRODUCE(x,y)
            if(small random prob.) then child ←
MUTATE(child)
            add child to new_population
    until an individual is fit enough, or enough time has elapsed
    return the best individual according to FITNESS
```

Fig. 1. The Genetic Algorithm Pseudocode [1].

D. Multiobjective Optimization

Timetabling is usually a multiobjective problem because there are many aspects that determine the quality of a schedule. For example, one aspect is that compulsory courses at the same level must not conflict with each other. Another aspect would be that lecturers cannot teach more than one course during the same time slot. Every constraint that determines whether a timetable is good or bad can be seen as one objective function. The problem is that sometimes not all objective functions can be maximized altogether. The factors that determine their value are highly interdependent. In particular, some objective functions can directly contradict each other, or at least form a tradeoff between them [3]. The goal of multiobjective optimization is then the maximization of all objective functions as optimally as is possible.

There are two general approaches in computing fitness function within a multiobjective optimization, i.e. aggregation-based and Pareto-based. An aggregation-based fitness value is obtained from the aggregation of all objective function values, or in other words, a linear combination of all objective functions. On the other hand, Pareto-based fitness uses the principle of domination to calculate the fitness value of an individual within a population. An individual A dominates individual B if all objective function values of A >= B. The purpose is to eliminate a bad individual (the dominated individual). An example of a Pareto-based algorithm is SPEA2.

E. SPEA 2

One of the best performing Pareto-based algorithms is Strength Pareto Optimization Algorithm (SPEA) 2. This algorithm maximizes all objective functions, no objective function is ignored. Therefore, this algorithm also maintains genetic diversity. This algorithm works with steps shown in Fig.2.

Input:	M (offspring population size) N (archive size) T (maximum number of generations)
Output:	A* (nondominated set)
Step 1:	Initialization: Generate an initial population P_0 and create the empty archive(external set) $A_0 = Ø$. Set t = 0.
Step 2:	Fitness assignment: Calculate fitness values of individuals in P_t and A_t.
Step 3:	Environtmental selection: Copy all nondominated individuals in P_t and A_t to A_{t+1}. If size of A_{t+1} exceeds N then reduce A_{t+1} by means of the truncation operator, otherwise if size of A_{t+1} is less than N then fill A_{t+1} with dominated individual in P_t and A_t.
Step 4:	Termination: If $t \geq T$ or another stopping criterion is satisfied then set A* to the set of decision vectors represented by the nondominated individuals in A_{t+1}. Stop.
Step 5:	Mating selection: Perform binary tournament selection with replacement on A_{t+1} in order to fill the mating pool.
Step 6:	Variation: Apply recombination and mutation operators to the mating pool and set P_{t+1} to the resulting population. Increment generation counter (t = t+1) and go to Step 2.

Fig. 2. The SPEA2 Algorithm [3].

III. Design of Timetabling as CSP

A. Variable and domain definitions

There are four elements that comprise a university timetable: courses, lecturers, classrooms, and time slots. Every course has the following information: (1) course name, (2) amount of credit units – referred to as SKS or *satuan kredit semester*, (3) course year/level, (4) a flag indicating whether it is compulsory, (5) number of course participants, and (6) prerequisite relationship with other courses. Every lecturer is associated with his/her (1) name, (2) expertise, (3) available time slots, and (4) maximum teaching load (in SKS). A classroom is associated with (1) classroom name/number and (2) capacity. Finally, each time slot is associated with (1) its day and time, and (2) a flag indicating whether it is before or after the lunch break.

At the Faculty of Computer Science, University of Indonesia case, timetabling can thus be represented as a CSP with variables $\{X_1, X_2, \ldots, X_n\}$ where n = total number of course units (SKS) offered during a semester. X_i is a 4-tuple $< c_i, l_i, r_i, s_i >$ where $1 \leq i \leq n$, c_i is a course, l_i is a lecturer, r_i is a classrooms, and s_i is a time slot. In other words, each individual credit unit (SKS) from course c_i is assigned to be taught by lecturer l_i in classroom r_i at time slot s_i.

A more precise mathematical elaboration is as follows. Given:

$C = \{Course_1, Course_2, ..., Course_p\}$
p = number of courses offered in a semester
$L = \{Lecturer_1, Lecturer_2, ..., Lecturer_q\}$
q = number of lecturers
$R = \{Room_1, Room_2, ..., Room_t\}$
t = number of rooms
$S = \{Slot_1, Slot_2, ..., Slot_u\}$
u = number of slots

the CSP variables are thus $\{X_1, X_2, \ldots, X_n\}$,
n = total number of SKS offered in one semester
$1 \leq i,j \leq n$; $i,j \in Integer$
$X_i = \{ (c_i, l_i, r_i, s_i) \mid c_i \in C, l_i \in L, r_i \in R, s_i \in S\}$

To illustrate this model, the following sample case is presented: a faculty called TinyFasilkom offers five courses during one particular semester. There are four lecturers and three classrooms in this faculty. Lecture activities only fall on Monday and Tuesday, thus there are a total of 16 time slots. Information about this TinyFasilkom faculty can be seen in Tables 1 to 4.

B. Constraints and fitness function design

A solution to a timetabling problem is the creation of a schedule that satisfies all constraints. In other words, all courses must be assigned lecturers, classrooms, and time slot configurations that satisfy all hard constraints and soft constraints. Hard constraints (HC) are constraints that must not be violated because if violated, the schedule becomes invalid, whereas soft constraints (SC) may be violated but ideally should be satisfied.

Hard constraints (HC):

1. The schedule assigned to all courses should be appropriate with their required teaching load. This constraint is called HC1. Assumption: a course load ranges between 1 to 4 SKS. The mathematical definition of this constraint:

$$\forall a \; sks(a) = \sum_{k=1}^{n} same(c_k, a), a \in C$$
$$sks(x) \in \{1,2,3,4\}, x \in C$$
$$same(x,y) \begin{cases} 1 \; if \; x = y \\ 0 \; if \; x \neq y \end{cases}$$

2. Compulsory courses at the same year/level must not overlap. This constraint is called HC2. Mathematical definition of this constraint:

$$\forall i, j \; i \neq j \wedge level(c_i) = level(c_j) \wedge comp(c_i) \wedge comp(c_j) \rightarrow s_i \neq s_j$$
$$level(x) \in \{1,2,3,4\}, x \in C$$
$$comp(x) \in \{true, \; false\}, x \in C$$

3. The same class room and time slot is not filled by more than one course. This constraint is called HC3. Mathematical definition of this constraint:

$$\forall i, j \; i \neq j \wedge (r_i \neq r_j \vee s_i \neq s_j)$$

TABLE I
COURSE INFORMATION FOR TINYFASILKOM

ID	Name	S	P	L	C
C0	DDP-A	4	60	1	Yes
C1	DDP-B	4	60	1	Yes
C2	PSD	4	120	1	Yes
C3	SDA	4	120	2	Yes
C4	OSK	3	60	2	Yes

Legend:
S = amount of credit units
P = number of course participants
L = course year/level
C = course is compulsory

TABLE II
LECTURER INFORMATION OF TINYFASILKOM

ID	Name	Expertise	Available time slots	Max Sks
L0	Halsen	[C0, C1]	[T0, T1, T2, T3, T8, T9, T10, T11]	8
L1	Kivin	[C2]	[T4, T5, T12, T13]	4
L2	Rus	[C3]	[T0, T1, T8, T9]	4
L3	Yuba	[C4]	[T4, T5, T12]	3

TABLE III
ROOMS INFORMATION OF TINYFASILKOM

Code	Name	Capacity
R0	R 2102	130
R1	R 2301	65
R2	R 2302	75

TABLE IV
SLOTS INFORMATION OF TINYFASILKOM

Code	Day Time	Before Lunch Break
T0	Monday (08.00-09.00)	Yes
T1	Monday (09.00-10.00)	Yes
T2	Monday (10.00-11.00)	Yes
T3	Monday (11.00-12.00)	Yes
T4	Monday (13.00-14.00)	No
T5	Monday (14.00-15.00)	No
T6	Monday (15.00-16.00)	No
T7	Monday (16.00-17.00)	No

4. A lecturer does not teach more than one course at the same time. This constraint is called HC4. Mathematical definition of this constraint:
$\forall i, j \; i \neq j \wedge (l_i \neq l_j \vee s_i \neq s_j)$
5. The number of participants of a course is smaller or equal to the assigned classroom capacity. This constraint is called HC5. Mathematical definition of this constraint:
$\forall i \; participant(c_i) \leq capacity(r_i)$
$participant(x) \in I, x \in C$
$capacity(x) \in I, x \in R$

Soft constraints (SC):

1. The teaching load of a lecturer does not exceed his/her maximum load. This constraint is called SC1. Mathematical definition of this constraint:
$\forall b \; load(b) = \sum_{k=1}^{n} same(l_k, b), b \in L$
$load(x) \in I, x \in L$
2. A lecturer only teaches courses which he/she possesses the required knowledge/expertise. This constraint is called SC2. Mathematical definition of this constraint:
$\forall i \; c_i \in \exp(l_i)$
$\exp(x) \subseteq C, x \in L$
3. A lecturer only teaches during his/her available time slot. This constraint is called SC3. Mathematical definition of this constraint:
$\forall i \; s_i \in available(l_i)$
$available(x) \subseteq S, x \in L$
4. Course schedules are not cut by the lunch break. This constraint is called SC4. Before presenting our mathematical modeling this constraint, we need to define some auxiliary operators: *timeslot*, which takes a course teaching assignment and returns the time slot in which it is scheduled, and *overlap*, which takes two time slots and indicates whether they overlap:
$slot(X) = \{s \mid s \in S \wedge X \text{ is taught during } s\}, x \in C$
$$overlap(sx, sy)\begin{cases} true \; if \; sy \subseteq sx \\ false \; if \; sx \subset sy \end{cases} sx, sy \in S$$
The mathematical definition of this constraint:
$\forall a, b \neg overlap(slot(a), b), a \in C, b \in V$
v = number of lunch break
$V = \{V_1, V_2, ..., V_v\}$
$V_c = \{x_c, y_c\}, x_c \neq y_c, x_c \in S, y_c \in S, 1 \leq c \leq v, c \in I$
5. Course schedules are not cut by day. This constraint is called SC5. Mathematical definition of this constraint:
$\forall a, b \neg overlap(slot(a), b), a \in C, b \in W$
w = number of day break
$W = \{W_1, W_2, ..., W_w\}$
$W_c = \{x_c, y_c\}, x_c \neq y_c, x_c \in S, y_c \in S, 1 \leq c \leq w, c \in I$

C. Chromosome Representation

To solve the timetabling problem using GAs we need to represent the schedule as a chromosome. There are many ways of achieving this. In this paper, four representations are explored. They are R1 ("CLRS"), R2 ("LRS"), R3 ("LRSRSF"), and R4 ("LRSRS"). CLRS representation is the canonical 'phenotypic' representation whereas the three other representations can be seen as higher level 'genotypic' representations that also encode some heuristics. Regarding the labels, the symbol C means courses, L means lecturers, R means rooms, S means slots, and F means flag.

Every symbol has its own domain. The domain of C is integer from 0 to number of courses – 1. The domain of L is integer from 0 to number of lecturers – 1. The domain of R is integer from 0 to number of rooms – 1. The domain of S is integer from 0 to number of slots – 1. Finally, the domain of F is 0 and 1.

R1 Representation (CLRS)

In this representation, every tuple $X_i = \{ (c_i, l_i, r_i, s_i) \mid 1 \leq i \leq n, i \in \text{integer}, c_i \in C, l_i \in L, r_i \in R, s_i \in S\}$ is explicitly stated as four integer values that represent the course code, lecturer code, room code, and slot code. Thus, the chromosome length in this representation is 4 x total SKS (n). If say there are 2 courses, each with a load of 3 SKS, then the total SKS is 2 x 3 = 6 and the chromosome length is 4 x 6 = 24. The CLRS representation is illustrated in Fig.3.

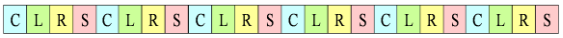

Fig. 3. CLRS representation.

If this representation is used on the TinyFasilkom sample above, the chromosome length will be 4 x 19 = 76. Illustration of CLRS representation for a complete assignment of the TinyFasilkom sample can be seen at Fig.4.

Fig. 4. CLRS representation in TinyFasilkom sample.

R2 Representation (LRS)

This representation is a modification of the canonical CLRS representation. In this representation, the course order and load is fixed. This also ensures that HC1 is satisfied. Chromosome length in this representation is

thus 3 x total SKS. If there are 2 courses, each with load of 3 SKS then the total SKS is 2 x 3 = 6 and the chromosome length is 3 x 6 = 18. The ordering is significant. There is a rule to determine which course corresponds to an LRS: the first course in *C* corresponds to the *m* first LRS, where *m* is the first course SKS load. Subsequently, the second course corresponds to the *n* next LRS, where *n* is the second course SKS load. The LRS representation illustration can be seen in Fig.5.

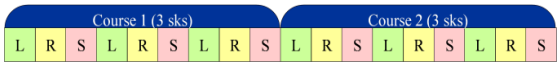

Fig. 5. LRS representation.

If this representation is used on the TinyFasilkom sample, the chromosome length will be 3 x 19 = 57. The illustration of LRS representation for a complete assignment of the TinyFasilkom sample can be seen at Fig.6.

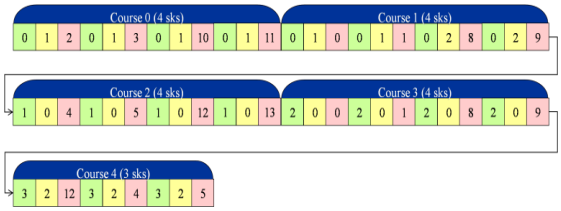

Fig. 6. LRS representation in TinyFasilkom sample.

R3 Representation (LRSRSF)

This representation is a modification from the LRS representation, thus HC1 will surely be satisfied. In this representation the lecture time of a course is divided into two meetings with the same lecturer, classroom, and time slot during each meeting. The chromosome length in this representation is 6 x the number of courses. If there are 2 courses, each with load 3 SKS then the chromosome length is 6 x 2 = 12. In this representation, a course is taught by only one lecturer and is divided into a maximum of two meetings, called the first meeting and second meeting. Here, flag (F) is functioned as a marker that indicates how the SKS load is distributed to these meetings. For courses with 3 SKS load, if F = 0 then 2 SKS is distributed to the first meeting and 1 SKS is distributed to the second meeting, whereas if F = 1 then 2 SKS is distributed to the second meeting and 1 SKS is distributed to the first meeting. For the case of courses with 1 or 2 SKS load, if F = 0 then only the first meeting occurs. Otherwise, if F = 1 then only second meeting happen. Finally, in a case of courses with 4 SKS load, both meetings will have 2 SKS load regardless of the value of F. LRSRSF representation illustration can be seen in Fig. 7.

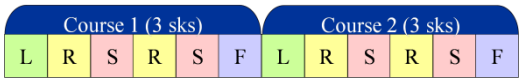

Fig. 7. LRSRSF representation.

If this representation is used on the TinyFasilkom sample, the chromosome length will be 6 x 5 = 30. Illustration of LRSRSF representation for a complete assignment of TinyFasilkom sample can be seen in Fig.8.

Fig. 8. LRSRSF representation in TinyFasilkom sample.

R4 Representation (LRSRS)

This representation is a slight modification of the R3 (LRSRSF) representation, where the F element is removed and F is thus always assumed to be 0.

IV. Experiment

Our experiments are conducted to determine the best variant of genetic algorithm and the best chromosome representation. We use a medium-sized test case taken from actual data in preceding semesters. We feel this test case represents a fairly realistic scenario at a typical small computer science department. Within this test case there are 20 courses to be taught. The resources within the faculty are: 20 lecturers, 17 classrooms, and 39 time slots. More specifically, the variables tested in this experiment are as follows:

1. Multiobjective algorithm: SPEA2 and *aggregation-based* (AB).
2. Representation: R1 (CLRS), R2 (LRS), R3 (LRSRSF), and R4 (LRSRS).

In our experiments, all combinations of the above multiobjective algorithms and representational schemes are tried. The GA parameters used are: population size 100, iteration number 5000, crossover rate 1, mutation rate 0.05, single-point crossover, archive size 50%. Each configuration is run with 5 different random seeds, and the presented results are the average over these 5 runs.

ECJ[1] is used for the implementation. The charts shown in Fig.9 and Fig.10 are convergence graphs which show the progression of the best fitness value within the population while the GA iteration occurs. The horizontal axis represents time/iteration from 0 –

[1] ECJ is a Java-based evolutionary computation library and can be downloaded from http://cs.gmu.edu/~eclab/projects/ecj/

5000 and the vertical axis represents the fitness value in the interval [0.0, 1.0]. The fitness value shown is the average of the best individual from the populations at each iteration from the 5 random seed runs. Fitness value calculation depends on the active constraints. The fitness value calculation for aggregation-based algorithm in this experiment uses the following fitness function formula, which is essentially a linear combination of the penalty contributions of each hard constraint (HC) and soft constraint (SC):

F(i)=(1/(1+HC1(i))+…+1/(1+HC5(i))+1/(1+SC1(i))+…+1/(1+SC5(i)))/10

The SPEA2 algorithm has its own way to calculate fitness value. However, this fitness value can not be compared with the aggregation-based algorithm fitness value. Therefore, the fitness value of individuals shown in Fig.10 is also calculated using the above aggregation-based fitness function formula while running SPEA2 algorithm. However, whilst the GA is running, it uses the domination-based SPEA2 fitness value.

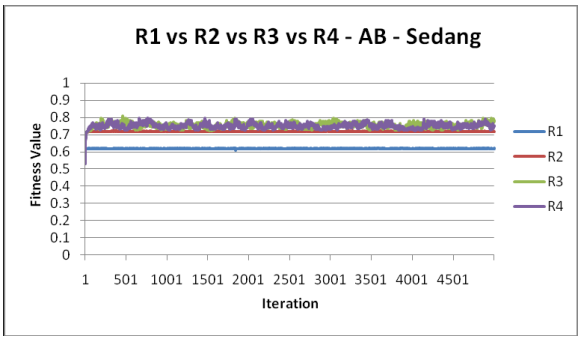

Fig. 9. R1 vs R2 vs R3 vs R4 - AB

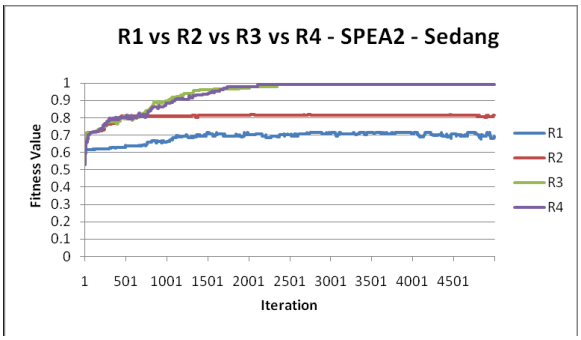

Fig. 10. R1 vs R2 vs R3 vs R4 – SPEA2

From Fig.9 we can see that R1, R2, R3, and R4 are unable to find an optimal solution. R3 and R4 only reach fitness value about 0.8, whereas R2 reaches a fitness value of 0.72, and R1 only reaches a fitness value of 0.62. However, in Fig.10, we can see that using R3 and R4 we are able to obtain an optimal solution. R4 is slightly faster than R3 as it takes 2200 iterations to converge at an optimal solution, whereas R3 takes 2400 iterations. R1 and R2 can only reach fitness values of 0.73 and 0.82 respectively. Thus we can conclude that the best representation is R4. Comparing between Fig.9 and Fig.10 we can also observe that the SPEA2 multiobjective algorithm performs better than the simple AB algorithm.

If we analyze the representational scheme, we can observe that R2 explores a larger search space than R3 and R4. This is due to the fact that R2 allows the possible state where a course is taught by more than one lecturer, whereas R3 and R4 enforces that a course can only be taught by one lecturer. In reality, no constraints actually prevent these states, and this makes R3 and R4 more bounded than R2. Intuitively we may expect that R3 and R4 prevent the GA from exploring potential solutions, but in fact the pruned search space enables it to find optimal solutions. According to [5], there are two ways that constraints can be implemented: ensuring that all possibly evolvable solutions never violate the constraint, or imposing penalties on individuals that violate a constraint. There is a trade-off: the former approach is obviously ideal, but its intractability is often the very reason GAs are employed in the first place. On the other hand, imposing excessively heavy penalties often leads to premature convergence on the first found well-formed solution, whereas if the penalties are too light, the GA may continue to evolve ill-formed solutions that score better than well-formed ones. Our experiments show that given the task of university timetabling, the balance achieved by representational schemes R3 and R4 is ideal.

V. Conclusion

The conclusion of this experiment is twofold: firstly, that the SPEA2 multiobjective algorithm is much more effective than the simple AB linear combination, and that secondly, the R3 and R4 representational scheme yields the best results.

References

[1] S. Russell and P. Norvig. (2003). Artificial Intelligence - A Modern Approach, 2nd edition, New Jersey: Prentice Hall.

[2] D.T. Soraya (2007). Penjadwalan Perkuliahan dengan pendekatan Constraint Programming, Fakultas Ilmu Komputer Universitas Indonesia.

[3] E. Zitzler, M. Laumanns, and S. Bleuler (2004). A Tutorial on Evolutionary Multiobjective Optimization, Swiss Federal Institute of Technology Zurich.

[4] T. Back, D. Fogel, and Z. Michalewicz, editors (1997). Handbook of Evolutionary Computation. Oxford University Press and Institute of Physics Publishing.

[5] D.E. Goldberg, (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, USA.