African Journal of Science and Technology (AJST)
Science and Engineering Series Vol. 7, No. 1, pp. 34 - 40

TABU SEARCH HEURISTIC FOR UNIVERSITY COURSE TIMETABLING PROBLEM

Mushi, A. R.

Department of Mathematics, University of Dar es salaam, Box 35062, Tanzania

ABSTRACT:- *In this study we have addressed the NP-Hard problem of academic course timetabling. This is the problem of assigning resources such as lecturers, rooms and courses to a fixed time period normally a week, while satisfying a number of problem-specific constraints. This paper describes a Tabu Search algorithm that creates timetables by heuristically minimizing penalties over infeasibilities. The algorithm is developed with special focus on the University of Dar-as-salaam and compares the results with a previous manually generated timetable. It has been found that, the Tabu Search technique gives better results given a careful selection of parameters.*

INTRODUCTION

Many types of timetabling problems exist, but most of them are NP-Hard (Cooper and Kingston, 1996) which create difficulties in obtaining workable timetables that meet all the user requirements. Timetabling problems can be found in employee allocation, transport networks, educational institutions and many industrial and sports activities. Specifically in higher educational institutions, the problem can be divided into course and examination timetabling. Our interest is on the course timetabling problem which essentially entails the assignment of courses, rooms, students and lecturers to a fixed time period, normally a working week, while satisfying a given number of constraints. These constraints can be divided into hard and soft. Hard constraints must be satisfied, while soft constraints are to be satisfied as much as possible. Timetabling problems are of much interest because of their real life applications. We address the course timetabling problem at the University of Dar es salaam (UDSM). Many approaches have been suggested in tackling this problem in other institutions including, Mathematical Programming (Werra 1985, Daskalaki et al 2004), Constraint logic programming (Abdennadher and Michael 1999, Panagiotis 1998), Graph Coloring (Miner et al, 1995), and many heuristic algorithms such as genetic algorithms (Corne et al 1993, Hiroaki et al 2004, Hitoshi et al 2002, Vestergaard 1997), Simulated Annealing (ElMohamed and Fox, 1998) and Tabu Search (White and Xie 2001). White et al, 2004, applied Tabu Search with longer term memory to a course timetabling problem, however their application did not consider room allocation in the optimization strategy. In our application, rooms are also a scarce resource and will have to be included in the optimization strategy. UDSM course timetabling involves a number of specific constraints which are not common to other Universities, which makes it an interesting problem to tackle.

Course timetabling at UDSM

UDSM is under a transformation process which involves among other issues, the expansion of student enrolment. Currently there are approximately 16,000 students in four campuses. Due to the expansion program, it was found necessary to have a central timetable office at each campus so as to optimize the use of the available resources. This paper focuses on course timetabling at the main campus which is the largest campus involving approximately 15,000 students. There are two semesters per academic year with approximately 750 courses in each semester, 106 rooms which include classrooms and laboratories, 650 lecturers, and 15,000 students to be scheduled on a five days week. Each day is made up of 13 one-hour time slots starting from 7.00 a.m. to 8 p.m. giving a total of 65 timeslots for the whole timetable period. No consideration is given to the lunch breaks except Fridays which are minimally used to allow for Muslims prayers.

The current practice is to use a ready-made software package which provides a set of tools that the timetable officer can use to simplify the work. The timetable is essentially created manually, using a set of tools that can help to detect collisions and suggest suitable timeslots. This is a long process and a semester timetable takes an average of three weeks to prepare given that all necessary data have been entered into the system. The necessary data includes the student registration details, lecturer-course assignments from departments, course requirements, and updated room capacities.

The main challenge is to automate the timetabling process and come up with a quick and optimized timetable. Due to unforeseen problems such as untimely data, it is not possible to ignore the manual process all together; the output of the automated system will however provide a highly advanced solution which can easily be modified by the manual systems afterwards. This automated system can also act as a very important 'what-if analysis' tool to administrators during decision making processes.

In this project, we use the following terminologies as applied to UDSM timetabling;

- Course – A set of subject content to be taught to a particular group of students.
- Event – An assignment of lecturer, room and a course to a one hour time interval. A course can have several events according to the number of hours set in the curriculum.
- Lecture – A set of events of the same course, which are required to be scheduled together in the same room and the same time. A lecture can have one or more events.
- Block – A lecture with more than one consecutive events.

The hard constraints are as follows;

1. No student can attend more than one lecture at the same time
2. No lecturer can teach more than one lecture at the same time
3. No room can occupy more than one lecture at the same time
4. No room can be assigned a lecture with more students than its capacity
5. Some courses are scheduled in blocks of more than one hour, these restrictions must be respected.

The soft constraints include;

1. As much as possible, minimize the use of early morning (7.00 a.m.) lunch hours (13-14) and late evening hours (18-20).
2. Specifically minimize the use of Friday 13-14 hour and 18-20 hour slots to allow for Muslim prayers and Adventists Sabbath day respectively.
3. Minimize continuous lectures/blocks of the same course in a day. It is preferred to spread them over the week as much as possible.
4. As much as possible, evening lectures starting from 18 to 20 hours should be assigned to rooms with standby generators so as to minimize loss of lecture hours due to frequent power cuts.
5. Minimize special preferences by lecturers, students and University administration.

Tabu Search Algorithm

Tabu Search is a global heuristic technique which tries to avoid falling into local optima by creating a special list called tabu (Reeves, 1993). Any solution which has been recently selected is put into a tabu list so that it becomes 'taboo' for a short period of time, depending on the length of the list. This minimises the chance of cycling in the same solution, and therefore create more chances of improvement by moving into un-explored areas of the search space. The work by Glover and Laguna 1997, gives a comprehensive description of the technique (see also Glover 1990). The algorithm applied in this project is as follows;

Tabu Search {
Initialize parameters;
Get Initial Solution (S_o);
Converged = false;
While Not Converged {
    Get a set of solutions S in neighborhood of S_o($S \in N(S_o)$)
    Moved = false;
    While Not Moved {
    if S Not Tabu {
        Given an objective function f, find $\sigma = f(S) - f(S_o)$;
        Push S into Tabu list
        Moved = true;
        if($\sigma < 0$) Accept solution ($S_o = S$);
        }
    Next $S \in N(S_o)$
    }
Converged = Test convergence;
}
return S_o as the best solution;
}

Representation of the problem

Given a total of *n* events, a timetable solution is represented as integer-valued matrix S_{nx3} of events such that for each event *e*, row 0 represents the course, row 1 is a timeslot and row 2 is a room assigned to event *e*. We have chosen this representation because it allows free representation of lectures of the same course without binding them to the particular course. This is essential since most courses have more than one lecture associated with them.

A course is associated with the number of students registered (course size), number of units and maximum number of consecutive hours per each lecture (block size). Units are used to determine the number of lecture hours to be scheduled per each course. According to UDSM curriculum, each course unit is equivalent to one lecture hour per week. Thus, a 5 unit's course with a maximum block of 2 hours per lecture will require scheduling of two 2-hour block of lecture sessions and an additional 1-hour lecture. This is represented by a matrix C_{mx3} of courses, where the 1st row stores size of the course, 2nd row keep units and 3rd row is for the maximum block size. Rooms are represented by an array whose index is room numbers and the content stores the room sizes.

Given a set of *m* courses, we define a conflict matrix M_{mxm} such that

$$M_{ij} = \begin{cases} 1 & \text{if course } i \text{ clashes with course } j \\ 0 & \textit{Otherwise} \end{cases}$$

Courses *i* and *j* clashes if they have at least one student or lecturer in common.

This is clearly a triangular matrix, which can be represented by a 1-dimensional array where the element (i,j) in lower triangle of M is mapped by a function

$$h(i,j) = \frac{(2n-j)(j-1)}{2} + i \text{ for all } i \text{ e" } j \text{ in the array.}$$

Checking for a possible student/lecturer collision between courses *i* and *j* simply involves finding the value of $h(i,j)$.

Cost function

Given a solution *s*, and a set of *k* constraints,

$$\text{minimize } f(s) = \sum_{i=1}^{k} \lambda_i f_i(s),$$

Each function f_i represents one of the constraints and each λ_i is the weight given to the constraint depending on its importance. The cost function includes both hard and soft constraints, but higher penalties are assigned to hard constraints to discourage them from selection.

Constraints

Noting the definition of solution *s* above, then;
s_{e1} = timeslot allocated to event *e*
s_{e0} = course allocated to event *e*
s_{e2} = room allocated to event *e*

Furthermore;

Let E = set of all timetable events

Then the components which makes up the cost function can be represented as follows;

i.) No student or lecturer can have two lectures at the same time.

Note that, courses assigned to two events *i* and *j* have a student/lecturer clash if they belong to the same timeslot (i.e. $s_{i1} = s_{j1}$) and $M_{ij} = 1$. Thus minimize $\lambda_1 f_1(s)$, where $f_1(s) = \sum_{\substack{(i,j)\in E \ni S_{i1}=S_{j1} \\ i<j}} M_{ij}$, and λ_1 is a sufficiently large value.

$f_1(s)$ is the total number of collisions associated with the current solution and a feasible solution must have $f_1(s) = 0$.

ii.) No more than one event can be assigned to the same room at the same timeslot. Two courses assigned to events (i, j) have room clashes if they have been scheduled in the same timeslot (i.e. $s_{i1}=s_{j1}$) and belong to the same room (i.e. $s_{i2}=s_{j2}$).

Let $a_{ij} = \begin{cases} 1 \text{ if } s_{i2} = s_{j2} \\ 0 \text{ Otherwise} \end{cases}$, and minimize $\lambda_2 f_2(s)$, where

$f_2(s) = \sum_{\substack{(i,j)\in E \ni s_{i1}=s_{j1} \\ i<j}} a_{ij}$.

In this case λ_2 is a sufficiently large value and $f_2(s)$ gives the total number of room clashes in the current solution. Obviously a feasible solution must have $f_2(s) = 0$.

iii.) No room can be assigned a course with more students than the room capacity. Going through all events E

in the solution structure, we calculate the number of times that a room has been assigned more students than its capacity.

Let *Cap* (*j*) = Capacity of object *j*, where *j* is either a course or a room.
Suppose $i = s_{eo}$, and $r = s_{e2}$, for an event *e* assigned to course s_{e0} and room s_{e2}, and define $b_{ir} = \begin{cases} 1 & if\ Cap(i) > Cap(r) \\ 0 & Otherwise \end{cases}$, for some course *i* and room *r*.

Then minimize $\lambda_3 f_3(s)$, where $f_3(s) = \sum_e \sum_{(i,r)\in E} b_{ir}$, and λ_3 is a sufficiently large value. $f_3(s)$ gives the total number of courses which have been assigned to rooms with lower capacity than the course requirement in the current solution. As in previous cases, $f_3(s) = 0$ is a necessary condition for feasibility.

iv.) Maximize the distance between two events or block of events of the same course.

Two events or block of events *i* and *j* belong to the same course if $s_{i0} = s_{j0}$. Thus for each pair of events (i, j) which belongs to the same course we would like to maximize distance between their timeslots i.e. maximize s_{i1}-s_{j1} for $i<j$. Since this is a minimization problem, we use the inverse square function. Thus minimize $\lambda_4 f_4(s)$, where $f_4(s) = \sum_{\substack{(i,j)\in E \ni s_{i0}=s_{j0} \\ i<j}} \frac{1}{(s_{i1} - s_{j1})^2}$, and λ_4 is significantly small compared to values in the hard constraints. Again the best possible value for $f_4(s)$ is 0.

v.) Minimal use of special times.

As much as possible, minimize the use of early morning (7.00 a.m.) lunch hours (13-14) and evening hours (18-20). Specifically we give higher penalty for the Friday 13-14 for Muslim prayers and 18-19 for Seventh day Adventists. These hours can easily be enumerated. For instance, Monday slots are 1, 7, 12, 13, and Tuesday slots are 14, 20, 26.

Let *H* = set of all special timeslots and minimize; $\lambda_5 f_5(s)$, where $f_5(s) = \sum_{e\in H} s_{e1}$. In this case, $f_5(s)$ gives the sum of all timeslots for courses which have been assigned to the less desired times. The weight given to λ_5 differs in value for different times in *H*. We give more weight to Friday Muslim and Adventists prayers, followed by lunch time hours. The best possible value of $f_5(s)$ is 0 i.e. when no examination violates the special times.

vi.) Minimal use of rooms with no standby generator in evenings.

Let G = set of all rooms fitted with standby generators and V = set of all evening (18-19) timeslots of the week.

Furthermore, let $\delta_e = \begin{cases} 1 \text{ if } s_{e1} \in V \wedge s_{e2} \notin G \\ 0 \qquad\qquad Otherwise \end{cases}$. Then minimize $\lambda_6 f_6(s)$,

where; $f_6(s) = \sum_e \delta_e$, and λ_6 is a weight value which is sufficient to minimize the use of evening rooms with no standby generators. The function $f_6(s)$ calculates the total number of events which have violated the standby generator constraint in the current solution *s*. When all examinations satisfy this constraint, we expect $f_6(s)$ to have a value of 0.

Initial timetable

It is important to have an easy and quick way of generating an initial timetable. We simply assign each event to the earliest possible feasible timeslot and earliest feasible room. To help reduce the risk of developing an infeasible solution, both courses and rooms are sorted in descending order of their sizes. The initial solution is to be feasible only by satisfying all hard constraints. Note that, it is not necessary though that initial solution should be completely feasible, since we penalize higher on hard constraints in the cost function. An initial solution is therefore preferred to be as feasible as possible, and any infeasibility can be tolerated in anticipation of improvement in the tabu search process.

Tabu variations

To get the best solution in a Tabu Search implementation, some variations may be necessary for particular problems. Many of these involves the types of possible move structures, aspiration criteria, varying the tabu list size, the use of longer memory, and modifications to the stopping criteria. This algorithm considered a number of important decisions on the original tabu search algorithm which are; selection of the type of moves, aspiration criteria, and stopping criteria.

Type of moves

Two types of moves have been tested on the algorithm. The first type of moves is as follows;

1. Select a random event *e* in the set of all possible events

2. Select randomly a new timeslot *t* in the set of all possible timeslots.
3. Assign the new timeslot *t* to the event at position *e*. If *e* is a member of a block of events, assign sequential timeslots including *t* to all members of the block.

This is identical to several moves described in literature (Thompson and Dowsland 1996, Reeves 1999). The size of the neighbourhood associated with this kind of move is $|N(s)| = |e| \times (|t|-1)$, where $|e|$ = total number of events, and $|t|$ = total number of timeslots.

The second type of moves is the swap of two event courses as follows;

1. Select randomly two event e_1, e_2 in the set of all possible events
2. Find whether it is possible to swap. It is possible to swap events only if they have the same block size.
3. If possible to swap, then swap course numbers of the two event blocks otherwise select another set of events and repeat 1 until swap is successful.

The size of the neighbourhood associated with this kind of move is $|N(s)| = |e| \times (|e|-1)$. This type of move affects both timeslots and rooms since a course is associated with both a timeslot and room number.

Aspiration criteria

Sometimes a candidate solution could be in the tabu list, but would bring large improvement in the solution if accepted. In this case an aspiration criterion is used, where a candidate solution with large improvement in the solution is accepted regardless of its tabu status. We accept any solution which brings an improvement with a value of

$$\sigma \leq -100$$

Stopping criteria

Fixed numbers of iterations have a disadvantage that, the algorithm can run for a long time without improvement just to complete the set number of iterations. We have used a stopping criterion which considers the number of iterations without a change in solution value. The algorithm stops after running 1000 iterations without solution change.

Summary of Results

The algorithm was tested on a course timetabling problem

Data	Value
Students	8161
Lecturers	607
Rooms	106
Courses	729
Total events	1570
Total timeslots	65

previously solved by manual methods on the 2003/2004 academic year for semester 1. A program is written in C++ and tests run on a 2.4GHz Pentium 4 processor. Table 1 shows data for the specific problem used in the test runs;

Table 1: Data for the tested problem at UDSM

The total number of students does not include first years since their number is not known at the time of timetable preparation. A single dummy student is used instead, to represent first years in each programme since they have similar core courses per programme. First years normally pick their convenient optional courses after the release of the timetable in case of any collision. The total number of events is the total number of timetable hours required for all courses in the semester. Table 2 shows the weights used in the cost function for each type of constraint.

Table 2: Weights used in the objective function

Weight	Value	Description
λ_1 - λ_3	100	Hard constraints
λ_4	10	Distance between events
λ_5	4	Friday Muslim prayers
	4	Seventh day Adventists
	2	Lunch times
	1	Morning and evening times
λ_6	3	Standby generators

These weights have been assigned according to our experience on the user needs. For instance, Friday prayers are considered to be more important than morning and evening time constraints, while standby generators in the evening rooms are less important compared to distance between events. Table 3 shows the performance of the algorithm for the two types of moves tested. The rows of

the table show the kind of constraints solved and their performance in the final solution. Both values are the average of performances for different randomly generated values using different seeds in the random number generator.

Table 3: Performance of the Algorithm

Initial cost: 3094.72		
	Time move	**Course swap**
Final Cost	1.74	1,390.80
Student/Lecturer Collisions	0	0
Room clashes	0	0
Room size	0	0
Event Distance	1.74	880
Special time penalties	0	510
Standby generators	0	0
Time (Seconds)	4,684.91	5,057.67
% Improvement	99%	55%

Clearly, time moves performs much better than course moves with an improvement of 99% from initial solution compared to 55% on the course swaps. This could stem from the fact that course moves are too restrictive on the possible set of moves, as these affects both time and room allocations. Since the best possible value for each function in the objective is zero, the best possible objective cost is zero. The quality of our algorithm is clearly very good as it closely approach the value of zero. The best solution was found after 4,684.91 seconds which is about 1 hour and 18 minutes. This time is quite tolerable in timetable applications. Figure 1 shows the performance improvement by iterations between the two types of moves. Clearly, time moves performs much better and shows a faster convergence to the best solution than course swaps.

We have also calculated the performance of the manual solution as it was generated and used in 2003/2004 for comparison with the automatically generated best solution. Table 4 is a summary of the performances in terms of constraint violations. Both cases were feasible by satisfying all hard constraints.

Table 4: Manual vs Automatic Performances

Constraints	Violations in Manual	Violations in automatic
Student/Lecturer Collision	0	0
Room clashes	0	0
Room size violations	0	0
Course event gaps	253.75	1.74
Special time penalties	659	0
Standby generators	0	0
Total violation cost	912.75	1.74
% Improvement from Co	71%	99%

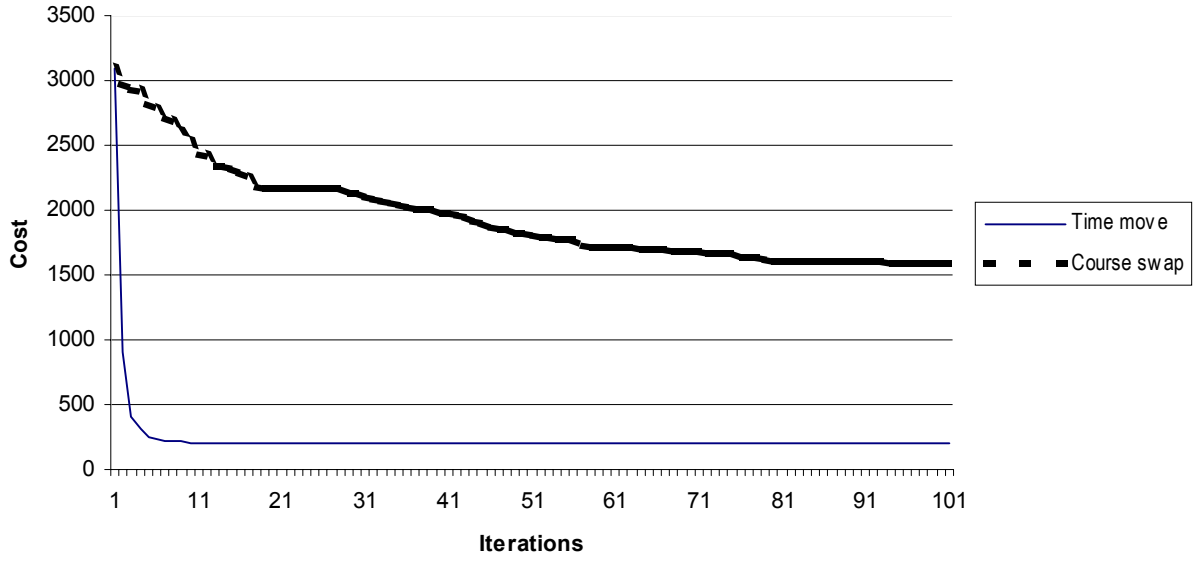

Figure 1: Performance improvement by iterations

The performance of manual solution is much lower (71%) compared to the automatically generated solution (99%). This is caused by more violations of the soft constraints in the manual than the automatic solution. The automatic system therefore performs better than the manual system.

CONCLUSION

The aim of the project was to develop a heuristic algorithm for the automatic generation of solution for the course timetabling problem at UDSM using tabu search technique. This has been achieved and demonstrated that tabu search is a good approach for the course timetabling problem at UDSM. Tabu search generally performs better than manually generated solution. It has also been found that the best move selection in this particular type of problem is the move of time slots. In addition, the use of aspiration and stopping criteria are important components in the success of the algorithm. Also tabu search heuristics are known to be dependent on careful selection of parameters. Therefore, further fine tuning of parameters might bring even better results.

REFERENCES

1. Abdennadher S, Michael M. (1999), University Timetabling using Constraint Handling rule, *Journal of Applied Artificial Intelligence*, Special issues on Constraint Handling Rules.
2. Cooper T., Kingston J. (1996). The Complexity of Timetable Construction Problems, *Springer Lecture Notes in Computer Science* 1153, pages 283-295
3. Corne D., Fang H., Mellis C. (1993), Solving the Modular Exam scheduling problem with Genetic Algorithms, *Proceedings of the sixth International Conference of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Edinburgh.
4. Daskalaki S., Birbas T., Housos E., (2004), An Integer Programming formulation for a case study in University timetabling, *European Journal of Operational Research*, Vol. 153, pp. 117-135.
5. ElMohamed M., Fox G. (1998), A Comparison of Annealing Techniques for Academic Course Scheduling, *Practice and Theory of Automated Timetabling II*, Selected Papers from the 2nd International Conference, PATAT'97, Edmund Burke and Mike Carter (Eds.), Lecture Notes in Computer Science, Springer
6. Glover, F. (1990), Tabu Search: A tutorial, *Interfaces* 20 (4), pp 74-94.
7. Glover, F., Laguna, M. (1997), Tabu Search, Kluwer Academic Publishers.
8. Hiroaki U., Daisuke O., Kenichi T., Tetsuhiro M. (2004), Comparisons of Genetic Algorithms for Timetabling Problems, *Systems and Computers in Japan*, Vol. 35, No. 7, pp 691-701.
9. Hitoshi K., Kondo M., Sugimoto M., (2002), Solving Timetabling Problems using Genetic Algorithms based on minimizing conflict heuristics, in Giannakoglou K., Tsahalis, Periaux, Papailiou, Fogrty(Eds), Evolutionary methods for design, Optimisation and Control, *CIMNE*, Barcelona.
10. Miner S., ElMohamed S., Hon W. (1995), Optimisation of Timetabling Solutions Using graph Colouring, *In Timetabling Techniques*, North-East Parallel Architecture Centre (NPAC).
11. Panagiotis S, Vigla E., Karaboyas F., (1998), Nearly Optimum Timetable Construction through CLP and Intelligent search, *International Journal on Artificial Intelligence Tools*, Vol. 7, No. 4, pp. 415-442.
12. Reeves C. (1993), Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publications, Oxford.
13. Reeves C. (1999). Landscapes. Operators and Heuristics Search, *Annals of Operations Research*, Vol. 86, pp. 473-490.
14. Thomspon J., Dowsland K. (1996). Variants of Simulated Annealing for the Examinations Timetabling Problem, *Annals of Operations Research*, Vol. 63, pp. 105-128.
15. Vestergaard L., (1997), Solving Timetabling Problem using Hybrid Genetic Algorithms, *Software – Practice and Experience*, Vol. 27(10), 1121-1134.
16. Werra, D. (1985). An Introduction to Timetabling, *European Journal of Operational Research,* 19, 151-162
17. White G., B. Xie (2001), Examination Timetabling and Tabu Search with longer term memory. In E. Burke and W. Erben, (Eds.), *The Practice and Theory of Automated Timetabling III*, Lecture Notes in Computer Science 2079, pages 85-103, Springer-Verlag.
18. White G.M., Xie S. B., Zonjic S. (2004), Using Tabu Search with longer term memory and relaxation to create examination timetables, *European Journal of Operations Research*, vol. 153, 80-91.