# Solving the University Class Scheduling Problem Using Advanced ILP Techniques

Ahmed Wasfy and Fadi A. Aloul

Department of Computer Engineering, American University of Sharjah (AUS), UAE
{*b00016690, faloul*}*@aus.edu*

***Abstract* — The University Class Scheduling Problem (UCSP) is concerned with assigning a number of courses to classrooms taking into consideration constraints like classroom capacities and university regulations. The problem also attempts to optimize the performance criteria and distribute the courses fairly to classrooms depending on the ratio of classroom capacities to course enrollments. The problem is a classical scheduling problem and considered to be NP-complete. It has received some research during the past few years given its wide use in colleges and universities. Several formulations and algorithms have been proposed to solve scheduling problems, most of which are based on local search techniques. In this paper, we propose a *complete* approach using integer linear programming (ILP) to solve the problem. The ILP model of interest is developed and solved using the three advanced ILP solvers based on generic algorithms and Boolean Satisfiability (SAT) techniques. SAT has been heavily researched in the past few years and has lead to the development of powerful 0-1 ILP solvers that can compete with the best available generic ILP solvers. Experimental results indicate that the proposed model is tractable for reasonable-sized UCSP problems.**

***Index Terms* — University Class Scheduling, Optimization, Integer Linear Programming (ILP), Boolean Satisfiability.**

## I. Introduction

The University Class Scheduling Problem (UCSP) represents an important class of optimization problems in operational research. It is considered one of the most difficult problems faced by universities and colleges today. Briefly defined, given a number of courses and classrooms, the goal is to assign courses to classrooms while satisfying all of the university constraints and optimizing the utilization of existing facilities effectively and efficiently.

Given the increasing number of students in universities, a large number of courses are offered every term. Each course has a different number of enrolled students and each classroom has different capacities which make the assignment of courses to classrooms complicated. Furthermore, it is not only enough to schedule a course in a classroom with a higher capacity than the number of enrolled students, since this can still lead to inefficient utilization of classrooms which can upset instructors and students. For-example, given two courses with 6 and 19 enrolled students, respectively, and two classrooms with capacities of 20 and 50 students, respectively, both courses can fit in either of the classrooms. However, it would make more sense to assign the larger course to the larger classroom which will improve the student's learning experience, allow additional students to attend the class, and reduce the chances of cheating when conducting exams.

Many formulations and algorithms have been proposed to solve UCSP. Most of these algorithms are based on *local* search techniques, namely hill climbing, simulated annealing, and tabu search [7][11][14][19]. Such algorithms cannot prove unsatisfiability or guarantee that a solution is optimal. In other words, if a solution is found, it cannot guarantee that this solution has the best possible optimization cost. In this paper, we propose an integer linear programming (ILP) approach to solve the UCSP. The approach is *complete* and hence examines the entire search space defined by the problem to prove that either (i) the problem has no solution, i.e. the problem is unsatisfiable, or (ii) that a solution does exist, i.e. the problem is satisfiable. If the problem is satisfiable, the proposed approach will search all possible solutions to find the optimal solution.

Recently, advanced Boolean Satisfiability (SAT) solvers have been extended to solve 0-1 ILP problems [1]. The SAT problem is a central problem in artificial intelligence and computer science and has received considerable attention from researchers. Many complex Engineering problems have been successfully solved using SAT. Such problems include routing [16], power optimization [2], verification [4], and graph coloring [17], etc. Today, several powerful SAT solvers exist and are able of handling problems consisting of thousands of variables and millions of constraints [10][13][15]. They can also compete with the best available generic ILP solvers.

In this paper, we will show how to formulate the UCSP as an ILP problem and study the possibility of solving the UCSP using (i) advanced SAT-based 0-1 ILP algorithms and (ii) generic-based ILP algorithms. We compare the performance of both algorithms and provide empirical results showing that generic-based ILP solvers tend to outperform SAT-based ILP solvers

for the proposed problem. The results include the actual schedule of classrooms and courses offered in the School of Engineering at the American University of Sharjah in addition to randomly generated set of courses and classrooms. The proposed approach is *complete* and is guaranteed to identify the optimal schedule.

This paper is organized as follows. Section 2 provides a general overview of SAT and ILP. Section 3 shows how to formulate the UCSP as a 0-1 ILP instance. A detailed example is shown in Section 4. Experimental results are presented and discussed in Section 5. The paper is concluded in Section 6.

## II. Boolean Satisfiability and Integer Linear Programming

Recent years have seen significant advances in Boolean satisfiability (SAT) solving. These advances have lead to the successful deployment of SAT solvers in a wide range of problems in Engineering and Computer Science. The SAT problem involves finding an assignment to a set of binary variables that satisfies a given set of constraints. In general, these constraints are expressed in *products-of-sum* form, also known as *conjunctive normal form* (CNF). A CNF formula $\varphi$ on $n$ binary variables $x_1,...,x_n$ consists of the conjunction (AND) of $m$ clauses $\omega_1,...,\omega_m$ each of which consists of the disjunction (OR) of literals. A literal is an occurrence of a Boolean variable or its complement.

As an example, the CNF instance:

$$f(a,b,c) = (a \vee b)(b \vee c)$$

consists of 3 variables, 2 clauses, and 4 literals. The assignment {$a$=1, $b$=0, $c$=0} leads to a conflict, whereas the assignment {$a$=1, $b$=0, $c$=1} satisfies $f$. Note that a problem with $n$ variables will have $2^n$ possible assignments to test. The above example with 3 variables has 8 possible assignments. An instance with 100 variables will have 1.27e+30 assignments. Assuming a processor that can verify an assignment every 1 nanosecond, the processor will complete testing all $2^{100}$ assignments in 4e+12 years.

Despite the SAT problem being NP-Complete [6], there have been dramatic improvements in SAT solver technology over the past decade. This has lead to the development of several powerful SAT algorithms that are capable of solving problems consisting of thousands of variables and millions of constraints. Such solvers include GRASP [13], zChaff [15], and Berkmin [10].

Most powerful SAT solvers are based on the original Davis-Putnam backtrack search algorithm [8]. The algorithm performs a depth first search process that traverses the space of $2^n$ variable assignments until a satisfying assignment is found (the formula is *satisfiable*), or all combinations have been exhausted (the formula is *unsatisfiable*). The search process proceeds as follows. Originally, all variables are unassigned. The algorithm begins by choosing a decision assignment to an unassigned variable. A decision tree is maintained to keep track of variable assignments. After each decision, the algorithm determines the implications of the assignment on other variables. This is obtained by forcing the assignment of the variable representing an unassigned literal in an unresolved clause, whose all other literals are assigned to 0, to satisfy the clause. This is referred to as the *unit clause* rule. If no conflict is detected, the algorithm makes a new decision on a new unassigned variable. Otherwise, the backtracking process un-assigns one or more recently assigned variables and the search continues in another area of the search space.

SAT solvers have been extended with several powerful algorithms to further expedite the search process. One of the best algorithms is known as the conflict analysis procedure [13] and has been implemented in almost all SAT solvers. Whenever a conflict is detected, the procedure identifies the causes of the conflict and augments the clause database with additional clauses, known as *conflict-induced clauses*, to avoid regenerating the same conflict in future parts of the search process. In essence, the procedure performs a form of learning from the encountered conflicts. Significant speedups have been achieved with the addition of conflict-induced clauses, as they tend to effectively prune the search space.

Intelligent decision heuristics and random restarts [15] also played an important role in enhancing the SAT solvers performance. The developers of the state-of-the-art SAT solver, Chaff [15], proposed an effective decision heuristic, known as VSIDS, and implemented several other enhancements, including random restarts, which lead to dramatic performance gains on many CNF instances.

Another recent extension to SAT solvers deals with its input format. Restricting the input of SAT solvers to CNF formulas can restrict their usage in various domains. Therefore, researchers have focused on extending SAT solvers to handle stronger input representations. Specifically, existing SAT solvers [1][9][18] have recently been extended to handle pseudo-Boolean (PB) constraints which are linear inequalities with integer coefficients that can be expressed in the normalized form [1] of:

$$a_1x_1 + a_2x_2 + ... + a_nx_n \geq b$$

where $a_i, b \in Z^+$ and $x_i$ are literals of Boolean variables. Note that any CNF clause can be viewed as a PB constraint, e.g. clause $(a \vee b \vee c)$ is equivalent to $a + b + c \geq 1$. PB constraints can, in some cases, replace an exponential number of CNF constraints. They have been found to be very efficient in expressing "counting constraints" [1]. Furthermore, PB extends SAT solvers

to handle *optimization* problems as opposed to only *decision* problems. This feature has introduced many new applications to the SAT domain. Specifically, all 0-1 ILP problems (i.e. ILP problems whose variables are Boolean) can be easily solved now by SAT solvers.

## III. Problem Formulation

In this paper, we are interested in evaluating the use of advanced ILP solvers, SAT- and generic-based, in solving the university class scheduling problem (UCSP). We start by describing how to formulate the problem as 0-1 ILP. To illustrate our approach, consider a university with *n* courses and *m* classrooms. In general, the number of classrooms *m* is greater than or equal to the number of courses *n*. For each course *i*, we define *m* variables as follows:

$x_{ij}$ = 1 if course *i* is assigned to classroom *j*,
0 otherwise.

Three sets of constraints are generated as follows:

- Each course must be assigned to one classroom. This can be expressed using the following PB constraint:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i$$

- Each classroom can fit up to one course to avoid scheduling two courses in the same classroom. This can be expressed using the following PB constraint:

$$\sum_{i=1}^{n} x_{ij} \le 1 \quad \forall j$$

- Each classroom capacity must be equal to or exceed the course's enrollment. This can be expressed using the following PB constraint:

$$\sum_{j \in T} x_{ij} = 0 \quad \forall i$$

Where *T* is the set of classrooms whose capacity is less than the number of enrolled students in class *i*.

The above three constraints will satisfy the university constraints and ensure that each course gets assigned to a classroom with a larger (or equal) capacity than the course's student enrollment. However, it will not optimize the utilization of the existing classrooms. It is still possible that large courses get assigned to small classrooms and small courses to large classrooms as shown in Section 1. In order to avoid such a scenario, and distribute the courses fairly and efficiently among the available classrooms, we add the following optimization PB objective function:

$$minimize\,(\sum c_{ij} x_{ij}) \quad \forall i, \forall j$$

where $c_{ij}$ is equal to the capacity of classroom *j* divided by the number of students enrolled in course *i*. The advantage of the objective function is described in Section IV.

By formulating the problem as such, we can do more than finding a fair schedule. We can incorporate any university restrictions or faculty preferences that we can think of in the resulting schedule. For example, by adding the PB constraint $x_{AZ} = 1$, we are forcing course *A* to be assigned to classroom *Z*. Similarly, we can exclude course *A* from being assigned to classroom *Y* by adding the PB constraint $x_{AY} = 1$.

We can also add dependencies between courses. For example, we can force one of two courses, e.g. *A* and *B*, to be assigned to a specific classroom *Z*. This can be expressed by adding the following CNF constraint $(x_{AZ} \vee x_{BZ})$.

We can also force certain courses to be assigned to specific classrooms only if a specific situation occurs. For example, we can force courses *A* and *B* to be assigned to classrooms *X* and *Z* only if course *C* is assigned to classroom *W*. This is expressed using the following set of CNF constraints $(\bar{x}_{CW} \vee x_{AX})$ and $(\bar{x}_{CW} \vee x_{BZ})$.

Note that the complexity of converting the class scheduling problem into a 0-1 ILP problem is $O(mn+k)$, where *k* is the number of course restrictions described above.

## IV. Illustrative Example

In this example, we will consider a university consisting of 3 classrooms (Class A, Class B and Class C), and offering 3 courses (Course 1, Course 2 and Course 3). Let's also assume the capacities and enrollments shown in Table 1.

TABLE I
Summary of class capacities and enrollments

| Class | Capacity | | Course | Enrollment |
|---|---|---|---|---|
| A | 10 | | 1 | 5 |
| B | 15 | | 2 | 18 |
| C | 20 | | 3 | 8 |

The problem will consist of 9 variables. The following constraints are generated:

- Each course must be assigned to one classroom

$$x_{1A} + x_{1B} + x_{1C} = 1$$
$$x_{2A} + x_{2B} + x_{2C} = 1$$
$$x_{3A} + x_{3B} + x_{3C} = 1$$

- Each classroom can fit up to one course

$$x_{1A} + x_{2A} + x_{3A} \le 1$$
$$x_{1B} + x_{2B} + x_{3B} \le 1$$
$$x_{1C} + x_{2C} + x_{3C} \le 1$$

- Classroom capacity must exceed course enrollment

$$x_{2A} + x_{2B} = 0$$

The optimization function is expressed as follows:

$$minimize \begin{pmatrix} 200x_{1A} + 56x_{2A} + 125x_{3A} + \\ 300x_{1B} + 83x_{2B} + 188x_{3B} + \\ 400x_{1C} + 111x_{2C} + 250x_{3C} \end{pmatrix}$$

Note that in the above expression, since the ILP solver can only accept integer coefficients, all ratios were multiplied by one hundred and were rounded to the nearest integer for simplicity.

In summary the example needs a total of 9 variables, 7 PB constraints, and 1 PB objective function. Two solutions are identified by the ILP solver and are summarized in Table 2. Clearly, the first solution is the best solution since it yields the smallest objective function value and assigns the smaller course, i.e. course 1, to the smaller classroom, i.e. classroom A, and the larger course, i.e. course 3, to the larger classroom, i.e. classroom C. Course 2 can only fit in classroom B.

TABLE II
Summary of identified solutions.

| **Solution #** | **Course Assignment** | | | **Optimization function value** |
|---|---|---|---|---|
| | **1** | **2** | **3** | |
| 1 | A | C | B | **499** |
| 2 | B | C | A | 536 |

## V. Experimental Results

In this section, we evaluate the use of ILP algorithms in solving the UCSP problem. A tool was developed using C# with an easy-to-use GUI interface. The user inputs the available courses and classrooms with their corresponding enrollment size and capacities, respectively. The tool also allows the user to upload an excel sheet that lists all available courses and classrooms to speed up and ease the data entry process. The tool then converts the user input into a 0-1 ILP instance as described in Section 3. The instance is then passed into a backend ILP solver and once the search is completed, the solution, if satisfiable, is converted into an easy-to-read schedule. The tool can handle an unlimited number of courses and classrooms, allows the user to enter unique preferences for each course or classroom, and is easily adaptable to use any 0-1 ILP solver in the backend as long as the solver accepts the input format generated by the tool. We used three of the best 0-1 ILP solvers: (1) The SAT-based 0-1 ILP solvers PBS [1][3] and MiniSAT+ [9], and (2) the leading commercial generic ILP solver, CPLEX [12]. PBS and MiniSAT+ are advanced SAT solvers that can handle both CNF and PB constraints. They employ the latest advances in the SAT technology. The experiments were tested on an Intel Xeon 3.2 GHz machine with 4 GB of RAM. The runtime limit was set to 1000 seconds. We present results for the Spring 2007 schedule of the School of Engineering at the American University of Sharjah (AUS). We also present results for 8 randomly generated sets of courses and classrooms. The random test case generator was modified to produce only satisfiable instances as following. For each case, the user would enter the number of courses $d$ and a minimum/maximum course enrollment size $g$/$h$. The generator defines a number of classrooms equal to the number of courses $d$ will be defined. For each course, the generator will set the enrollment size to a random value $q$ that is $g \le q \le h$. A corresponding classroom will be assigned a random capacity $r$ such that $q \le r \le h$.

Table III shows the results using AUS's course schedule. The course slots are shown for each case. A UTR slot means that the course is taught on Sundays, Tuesdays, and Thursdays. Similarly, a MW slot means that the course is taught on Mondays and Wednesdays. The table also shows the course timings. A single ILP instance is generated for all courses per time slot. The table shows the results obtained by the ILP solvers. Specifically, the table shows the runtime in seconds (Time), the value of the optimization objective function (Weight), and whether the solution is optimal or not (Opt). If the solver times-out, i.e. is unable to complete the search within the 1000 seconds runtime limit, we report the best found solution. All solvers should yield the same optimal solution if they complete the search. Finally, in order to evaluate the advantage of the proposed approach, we report in the third column of the table the value of the objective function using the current AUS class assignment.

Table IV shows the results using the randomly-generated course schedules. The table shows a description of courses and classrooms size and the results obtained by the ILP solvers. Several observations are in order:

- For the AUS schedule, all solvers identify better schedules than the existing course schedule used at AUS for all time slots. In fact, the schedule identified by CPLEX has a weight that is on average 25% smaller than the weight identified by the existing schedule.
- In some cases, CPLEX was able to improve the weight by a factor of almost 40% as in the MW 11-12:15 case.
- The generic ILP solver CPLEX successfully solves and identifies the optimal solution for all instances in less than a second.
- The SAT-based ILP solvers, PBS and MiniSAT+, time-out in all but one case, but return the best found solution. Giving the solvers extra time would have probably help find a better solution or even solve the problem by finding the optimal solution.
- The generic ILP solver, CPLEX, outperforms the SAT-based ILP solvers for the proposed problem.
- The larger the instance, i.e. number of classes and classrooms, the longer is the search runtime.
- The approach is complete and is guaranteed to find a fair schedule given enough time and memory resources.

TABLE III

SUMMARY OF RESULTS USING THE AUS SCHEDULE.

| Test Case | AUS School of Engineering Schedule | | | MiniSAT+ | | | PBS | | | CPLEX | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Class Slot | # of Courses | Weight | Time | Opt | Weight | Time | Opt | Weight | Time | Opt | Weight |
| 1 | UTR 8-8:50 | 10 | 2052 | >1K | No | 1436 | >1K | No | 1444 | **0** | Yes | 1436 |
| 2 | UTR 9-9:50 | 12 | 2017 | >1K | No | 1687 | >1K | No | 1689 | **0** | Yes | 1686 |
| 3 | UTR 10-10:50 | 10 | 1938 | >1K | No | 1511 | >1K | No | 1522 | **0** | Yes | 1511 |
| 4 | UTR 11-11:50 | 12 | 2288 | >1K | No | 1938 | >1K | No | 1940 | **0** | Yes | 1936 |
| 5 | UTR 12-12:50 | 11 | 1932 | >1K | No | 1630 | >1K | No | 1638 | **0.01** | Yes | 1627 |
| 6 | UTR 1-1:50 | 11 | 2173 | >1K | No | 1608 | >1K | No | 1613 | **0** | Yes | 1607 |
| 7 | MW 8-9:15 | 10 | 1721 | >1K | No | 1577 | >1K | No | 1592 | **0** | Yes | 1574 |
| 8 | MW 9:30-10:45 | 10 | 1923 | >1K | No | 1413 | >1K | No | 1426 | **0.01** | Yes | 1413 |
| 9 | MW 11-12:15 | 3 | 616 | **0.14** | Yes | 366 | **0.01** | Yes | 366 | **0** | Yes | 366 |
| 10 | MW 12:30-1:45 | 9 | 1628 | 384 | Yes | 1274 | >1K | No | 1277 | **0.01** | Yes | 1274 |

TABLE IV

SUMMARY OF RESULTS USING THE RANDOMLY GENERATED SCHEDULE.

| Test Case | # of Courses | Enrollment | | MiniSAT+ | | | PBS | | | CPLEX | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Min | Max | Time | Opt | Weight | Time | Opt | Weight | Time | Opt | Weight |
| 1 | 20 | 10 | 25 | >1K | No | 2392 | >1K | No | 2391 | **0.01** | Yes | 2385 |
| 2 | 20 | 10 | 70 | >1K | No | 2957 | >1K | No | 2933 | **0.01** | Yes | 2925 |
| 3 | 40 | 10 | 25 | >1K | No | 4827 | >1K | No | 4810 | **0.02** | Yes | 4786 |
| 4 | 40 | 10 | 70 | >1K | No | 5882 | >1K | No | 5866 | **0.02** | Yes | 5762 |
| 5 | 60 | 10 | 25 | >1K | No | 7610 | >1K | No | 7586 | **0.06** | Yes | 7523 |
| 6 | 60 | 10 | 70 | >1K | No | 10324 | >1K | No | 10311 | **0.03** | Yes | 9928 |
| 7 | 80 | 10 | 70 | >1K | No | 12930 | >1K | No | 12735 | **0.17** | Yes | 12302 |
| 8 | 100 | 10 | 70 | >1K | No | 17437 | >1K | No | 17186 | **0.3** | Yes | 16307 |

## VI. CONCLUSIONS

In this paper, we present an ILP-based approach to generating university course schedules. We show how to formulate the university schedule as an ILP problem. The ILP models are solved using advanced generic-based and Boolean satisfiability based ILP solvers. The goal is to find a schedule that satisfies the university's rules, yet optimizes the use of the existing facilities such as minimizing the capacity to enrollment ratio in a class. The approach was tested on different cases with various sizes and showed promising results. The approach is *complete* and will find the best possible schedule, or will indicate that no schedule exists that meets the current university rules.

## REFERENCES


[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Generic ILP Versus Specialized 0-1 ILP: An Update," in *Proc. of the Int'l Conference on Computer Aided Design*, 450-457, 2002.

[2] F. Aloul, S. Hassoun, K. Sakallah, and D. Blaauw, "Robust SAT-Based Search Algorithm for Leakage Power Reduction," in *Proc. of the Int'l Workshop on Power and Timing Modeling, Optimization, and Simulation* (PATMOS), 167-177, 2002.

[3] B. Al-Rawi and F. Aloul, PBS4 SAT Solver. Available at: *http://www.eecs.umich.edu/~faloul/Tools/pbs4*

[4] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking Using SAT Procedures Instead of BDDs," in *Proc. of the Design Automation Conf.*, 317-320, 1999.

[5] D. Chai and A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver", in *Proc. of the Design Automation Conference* (DAC), 830-835, 2003.

[6] S. Cook, "The Complexity of Theorem Proving Procedures," in *Proc. ACM Symp. on the Theory of Computing*, 151-158, 2004.

[7] D. Cosla, "A Tabu Search Algorithm for Computing an Operational Timetable," in *European Journal of Operational Research*, vol. 76, 98-110, 1994.

[8] M. Davis, G. Longman, and D. Loveland, "A Machine Program for Theorem Proving," in *J. of the ACM*. 5(7), 394-397, 1962.

[9] N. Een and N. Sorensson, "An Extensible SAT-solver," in *Proc. of the International Conference on Theory and Applications of Satisfiability* Testing (SAT), 502-508, 2003.

[10] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust SAT-solver," in *Proc. of the Design Automation and Test Conference in Europe* (DATE), 142-149, 2002.

[11] A. Hertz, "Tabu Search for Large Scale Timetabling Problems," in *European Journal of Operation Research*, 54(1), 39-47, 1991.

[12] ILOG CPLEX, http://www.ilog.com/products/cplex

[13] J. Marques-Silva and K. Sakallah "GRASP: A Search Algorithm for Propositional Satisfiability," in *IEEE Trans. On Computers*, 48(5), 506-521, 1999.

[14] E. Mooney, R. Dargen, and W. Parameter, "Large-Scale Classroom Scheduling," in *IIE Trans.*, 28(5), 369-378, 1996.

[15] K. Moskewicz, C. Madigan, Y. Zho, and S. Malik "Chaff: Engineering an efficient SAT solver," in *Proc. of the Design Automation Conference* (DAC), 503-535, 2001.

[16] G. Nam, F. A. Aloul, K. A. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," in *Proc. of the International Symposium on Physical Design* (ISPD), 222-227, 2001.

[17] A. Ramani, F. Aloul, I. Markov, and K. Sakallah, "Breaking Instance-Independent Symmetries in Exact Graph Coloring," in *J. of Artificial Intelligence Research*, vol. 26, 289-322, 2006.

[18] H. Sheini and K. Sakallah, "Pueblo: A Modern Pseudo- Boolean SAT Solver," in *Proc. of the Design, Automation, and Test Conference in Europe* (DATE), vol. 2, 684-685, 2005.

[19] V. Tam and D. Ting, "Combining the Min-Conflicts and Look-Forward Heuristics to Effectively Solve a Set of Hard University Timetabling Problems," in *Proc. of the IEEE International Conference on Tools with Artificial Intelligence*, 2003.