5-1-2014

Finding Maximum Subsets of Relational Objects using SAT Decomposition

Bryan Pearce
Clemson University

Mary E. Kurz
Clemson University

Kavit Antani
Clemson University

Laine Mears
Clemson University, mears@clemson.edu

Proceedings of the 2014 Industrial and Systems Engineering Research Conference
Y. Guan and H. Liao, eds.

Finding Maximum Subsets of Relational Objects using SAT Decomposition

Bryan Pearce, Mary E. Kurz
Department of Industrial Engineering, Clemson University
Clemson, SC 29634, United States

Kavit Antani, Laine Mears
Department of Automotive Engineering, Clemson University
Greenville, SC 29607, United States

Abstract

Mass customization production modeling and analysis methods classically rely on explicit specification of the set of unique configurations possible. Growth in the number of available consumer options presents a challenge to this paradigm by combinatorial growth in the enumeration of the configuration set. In an alternative paradigm, uniquely configured products are defined implicitly by a database of object relation rules. We present an algorithm for finding the maximum bound on cumulative object attributes, aggregated over an explicitly defined set of objects. The algorithm searches for the largest valid subset using a Boolean encoding scheme and decomposing the problem into a series of smaller instances of the Boolean satisfiability problem (SAT). An assembly line balancing example is given as motivation for this problem, with task times as the object attributes in question. For this example, the algorithm finds bounds on the total time requirement of a set of tasks that are subject to object relations.

Keywords
Industrial engineering, operations research, mass customization, line balancing, Boolean satisfiability

Introduction
Mass customization systems manufacture variations of a common base product that differ according to a set of customizable options [1], and have been subject to a massive amount of production planning in recent decades. The *model-mix* paradigm is nearly universal within this literature, in which a *model* consists of all output products with identical customization attributes. Each unique model may be considered as a batch of identical product with individual and independent production volume, resource usage, and other problem variables. Models are largely independent from one another, as there is little interaction between models in many production environments. Inter-model setup times are the notable exception, and the lot-sizing and scheduling modeling methods in such environments are typically very focused on this interaction.
As the number of configurable options increases, the size of the model set grows at a combinatorial rate in response resulting to several difficulties when applying the model-mix paradigm in environments with a large number of options. For example, [2] discusses a modern automobile assembly line that features on the order of 10^{32} unique models. At this scale computational methods that iterate over the model set will be faced with exceptional memory and processing time requirements. Further, it may prove impossible to collect the necessary input data for each model. For example, demand for each model is very difficult to estimate when the model mix is many orders of magnitude larger than production volume.
The *option-mix* paradigm, as discussed by [3] offers an alternative information model in which individual variables are assigned for each option rather than for each model. Reliable estimates of option-mix frequencies remain feasible even with high product variety, e.g. the fraction of cars with optional heads-up display systems. The primary difficulty with the options-mix paradigm is that options are not direct abstractions of production units, as models are. Each production unit may possess zero, one, or many options, according to its configuration. Moreover, options may exhibit strong interaction with each other, in contrast with the relatively weak interaction between models noted above. Consider the case of stereo speakers in a vehicle: while basic, premium, and perhaps several other types of speaker options may be available, only one of these options may be installed in any given

vehicle. Such interaction information is necessary to options-mix methods, and herein we will assume that a rules database exists that documents these interactions. Such databases usually are maintained by product design or marketing departments within the organization, and facilitate the translation of options into feasible product configurations.

We would like to note that while the options-mix paradigm can be applied to any problem in the model-mix domain, each problem presents unique challenges. Herein we will devote attention to an assembly line balancing (ALB) problem in which assembly tasks may be associated with a large number of optional parts. In Section 1 the ALB problem is introduced, along with the horizontal balancing problem that motivates the options-mix problem. In Section 2 the data environment that comprises the options-mix information model for this problem is presented. Section 3 describes the application of instances of the Boolean satisfiability problem to evaluate a metric for the horizontal line balancing problem in this environment.

1. Assembly Line Balancing

The traditional form of an assembly line, as described by [4], is a production system consisting of a configuration of consecutive workstations, typically using a conveyor or similar to transport production units down the line. The total work to be performed along the assembly line is subdivided into the smallest indivisible elements of work, called tasks, each of which possesses an associated task time (t_i). Tasks are related to one another by precedence attributes, i.e. some tasks must be finished before others can begin, usually due to the physical architecture of the product. These individual precedence relationships between tasks are collected and summarized by a precedence graph, an acyclic graph with each task as a node and arcs representing precedence. Stations are spaced along the line such that there is one production unit present at each station, and all stations are allotted a fixed cycle time (C) to execute all assigned tasks before the conveyor moves the product to the next station. The ALB problem is to assign the set of tasks to stations, such that all work is performed upon the product as it traverses the line.

Assembly lines were originally constructed for mass production of standardized assembly products, to increase average worker productivity and overall throughput by leveraging labor specialization along the line [5]. Modern assembly lines designed for make-to-order and mass customization production permit fast and flexible responses to customer demand [6], but are associated with significant automation and facility capital costs. See [7] for a recent survey of modern mixed-model methods.

Tool setup times and the subsequent lot sizing problem are typically avoided by assembly lines via application of universal machinery or the like, in order to maintain consistent flow of production cycles [8]. Lines that do require setup time to transition from one model to another are referred to as multi-model lines, and encourage batches of each model to be produced consecutively. Such lines require an additional lot sizing problem extension, as discussed by [9,10].

1.1 Horizontal Line Balancing

Optimization of the traditional ALB problem seeks to minimize total idle time by minimizing of the number of stations (or workers) used, given a fixed cycle time. The problem is NP-hard, as shown by [11]. [12,13] transformed the mixed-model problem into a single model version by taking the demand-averaged time for each task. Such methods ignore the piece-to-piece variability in work content, and may result in disruptions in line operation. [12] attempted to compensate for this effect by minimizing a secondary objective of the sum of absolute deviations of actual station times of each model to the average station time across models, an early form of horizontal balancing. Horizontal balancing seeks to equalize the work content at a station across all model alternatives, such that the resulting balance is more robust to changes in model demand and production sequencing. [14] proposed a refined horizontal balancing objective that seeks to minimize the sum of work overload time, i.e. the work content in excess of the cycle time, across all models and stations. [15] developed a simulated annealing solution approach that incorporated both horizontal and vertical balancing objectives, within a model with parallel stations and additional assignment constraints.

The production sequencing problem that emerges from mixed-model environments can be solved in a staged fashion, subsequent to the ALB problem [16-19], or the two problems can be solved simultaneously [20]. Demand is commonly realized at a shorter time horizon than is applicable to the ALB problem, however, suggesting that ALB methods that produce solutions that are robust to demand may be more applicable than those that simultaneously sequence the production units. To test the effectiveness of different horizontal line balancing metrics to this end, [21] conducted extensive computational experiments to detect differences in line disruption due to product variety on assembly lines balanced with an array of different horizontal metrics. [21] also mentions in closure the reliance of all tested methods on the mixed-model paradigm, and calls for methods that are robust to product variety size.

2. Data Environment

The necessary data inputs to the options-mix horizontal line balancing problem are three-fold: 1) the set of tasks (denoted I) and associated task attributes, 2) the set of options (denoted Ω) and associated attributes, and 3) the database of object relations that defines option interactions. Before developing the attributes of these data inputs further it will be necessary to introduce the concept of a *platform*.

2.1 Platforms

It is assumed that each production unit is assembled to the specification of a single platform, selected from the set of platforms (denoted Ψ). Each platform represents a partially configured product, unique in name, to assist both customers and internal operations in differentiating the vast array of product configurations into a more manageable subset of categories. For customers, specification of product configuration begins with selection of platform, which serves to assign a default subset of high-profile option content to the product while leaving low-profile options undetermined. As an example, customer choice of the motorsports platform preselects engine, transmission, drivetrain, and brakes, while reducing the subset of available options for wheels and external trim. Full product configuration is then completed by the customer via specifying the remaining subset of options.

This approach affords the organization a tighter control of branding via the offering of several distinctive platforms, rather than a single amorphous product. An addition organizational benefit is reduced effort in creation and maintenance of the database of object relations.

Options are linked directly to each platform category with object relation flags. Selection of a platform may either require the option, forbid the option, or leave the option undetermined. See Table 1 for an example of how object relations between options and platforms is recorded in the database.

Table 1: Object relations between options and platforms. (M = mandatory, F = forbidden, O = optional)

	Platform 1	Platform 2	Platform 3
Option 1	M	M	F
Option 2	F	M	F
Option 3	O	O	O

Further control of relations between undetermined options is achieved via Boolean statements of the rule to be enforced within the database. Each rule is flagged for the platforms toward which it applies, and must be obeyed for products within the flagged platforms. For example, to enforce the rule "IF option 1 THEN NOT option 2" for platforms 1 and 2, but not platform 3, then the database would encode the rule as seen in Table 2.

Table 2: Object relations between rules and platforms. (T = rule applies to platform)

	Platform 1	Platform 2	Platform 3
IF option 1 THEN NOT option 2	T	T	

Each task is also flagged for the platforms towards which it is necessary. Task applicability to platforms are encoded into one of three classifications: 1) the task is applied universally across all platforms, 2) the task will only apply to a subset of the platforms, but will be necessary for all production units within the subset, and 3) the task applies to a subset of platforms, but only if some option is present. Table 3 presents an example of this encoding scheme in which task 1 is universal, task 2 is dependent on platform, and task 3 is dependent on both platform and option. The code word SERIES indicates that a task applies to all production units in the flagged platforms.

Table 3: Object relations between tasks and platforms. (A = Task applies to platform if condition is met.)

	Condition	Platform 1	Platform 2	Platform 3
Task 1	SERIES	A	A	A
Task 2	SERIES		A	
Task 3	Option 1	A	A	

2.2 Demand

Demand is specified on the platform level as the relative frequency of each platform $d_\psi, \psi \in \Psi$. Platforms are mutually exclusive to one another, and each product must be of a single platform, hence the total probability across all platforms is equal to one, $\sum_\Psi d_\psi = 1$. This characterization of demand alone is insufficient, however, as it does

not capture the demand of options that at not determined by selection of platform. To completely specify demand it is necessary to also introduce the probability of each option conditioned upon each platform: $d_{\omega,\psi} = P(\omega|\psi), \forall\ \omega \in \Omega, \psi \in \Psi$. Of course, forbidden option / platform combinations result in $d_{\omega,\psi} = 0$, and mandatory combinations have $d_{\omega,\psi} = 1$. It is only optional combinations that must be specified by the input data.

The proportional demand for each task, d_i, can be derived as a function of the $d_{\omega,\psi}$ and d_ψ data, by application of the equation

$$d_i = \sum_{\psi \in \Psi} G1_{i,\psi} d_\psi + \sum_{\omega \in \Omega, \psi \in \Psi} G2_{i,\omega,\psi} d_{\omega,\psi} \qquad \forall i \in I,$$

where $G1_{i,\psi}$ is an indicator variable equal to 1 if task *i* is SERIES for platform ψ and 0 otherwise, and $G2_{i,\omega,\psi}$is an indicator variable equal to 1 if task *i* is requires option ω for platform ψ and 0 otherwise.

3. Maximum Bound on Cycle Time

Using the data inputs described in Section 2, we now have sufficient information to derive a horizontal line balancing metric for the options-mix ALB. Given a line balancing solution with all tasks assigned to *M* stations, the average utilization of station *m* is calculated by considering the subset of tasks assigned to station *m*, I_m, and then taking the weighted average of task times proportional to the cycle time:

$$Utilization_m = \sum_{i \epsilon I_m} \frac{d_i t_i}{C}$$

Utilization is a valuable metric for the classic ALB problem, and must be restricted to be $\leq 1$for every station for the solution to be feasible to the cycle time. However, product variety may result in variability in the realized time usage from one product to the next. In the following we develop a method to calculate the maximum amount of time that might be required of station *m* to complete its assigned tasks. This upper bound is not necessarily simply the sum of all task times. If some tasks are linked to options content then it is possible that rules within the object relation database forbid execution of all tasks on any single product. Evaluation whether a subset of tasks may co-exist upon any single product is achieved by first parsing all rules in the database into a Boolean encoding scheme, and then solving the resultant Boolean satisfiability problem (SAT). The SAT problem considers a given set of primitives that are related by a given set of Boolean statements and determines whether there is any possible combination of true/false values that may be assigned to the primitives such that all statements are true. In Section 3.1 we show how to derive Boolean statements from the configuration data of the line balancing problem, and then parse those statements into binary parse trees. In Section 3.2 we show how to output the binary trees to a SAT solver to determine which tasks assigned to a station may occur simultaneously on some unknown product to be assembled.

3.1 Logical Statement Construction

Before the SAT problem it is first necessary to translate all of the configuration information from database fields and rule strings show in Section 2 into a set of Boolean statements. Considering the example data provided in Table 1 above, equivalent Boolean statements are:

1. (IF platform1 THEN option1) AND (IF platform2 THEN option1) AND (IF platform3 THEN NOT option1)
2. (IF platform1 THEN NOT option2) AND (IF platform2 THEN option2) AND (IF platform3 THEN NOT option2)

Notice each platform is treated with a separate IF clause, and the all of the platform clauses are joined with AND conjunctions. No information within Table 1 relate option3 with any of the platforms, and thus no Boolean statement is made regarding option3 as a result.

Considering the example data provided in Table 2, the equivalent Boolean statement is:

1) (IF platform1 THEN (IF option1 THEN NOT option2)) AND (IF platform2 THEN (IF option1 THEN NOT option2))

Considering the example data provided in Table 3 above, equivalent Boolean statements are:

1) IF platform1 THEN (task1 AND NOT task2 AND (IF option1 THEN task3))
2) IF platform2 THEN (task1 AND task2 AND (IF option1 THEN task3))
3) IF platform3 THEN (task1 AND NOT task2 AND NOT task3)

Next it is necessary to add rules dictating that a product can be of one and only one platform. Commonly referred to as "pick one" or "one hot lead" conditions, such rules are difficult to express using Boolean algebra expressions. It

is sufficient to enumerate these rules pairwise, e.g. "IF platform1 THEN NOT platform2," and so on. Although the number of rules required to enforce this constraint grows combinatorially with respect to the number of platforms, at the scale of this example problem (~20 platforms) the size of this rule set is still feasible.

A binary parse tree is then used to translate each derived Boolean statement string into a more manageable data structure via depth-first recursive parsing of the strings. Within the tree structure each node is a logical operator or a primitive, with the following node types being sufficient to encapsulate the information: AND, OR, NOT, IF, and PRIMITIVE. Each node may possess up to two children nodes, as needed to complete the logical operator. PRIMITIVES are the codes that represent options, platforms, or tasks, and are Boolean values that can be assigned true or false. PRIMITIVES exist only in leaf nodes and comprise all leaf nodes. Table 4 summarizes each of the node types within the parse tree.

Table 4: Nodes in binary parse tree

Node type	# Children	Description
AND	2	Both children evaluate to TRUE
OR	2	Either child must evaluate to TRUE
NOT	1	Child must be FALSE
IF	2	If left child is TRUE, then right child must be TRUE
PRIMITIVE	0	Object (option, platform, or task)

Figure 1 shows an illustration of the binary tree representation of a Boolean statement.

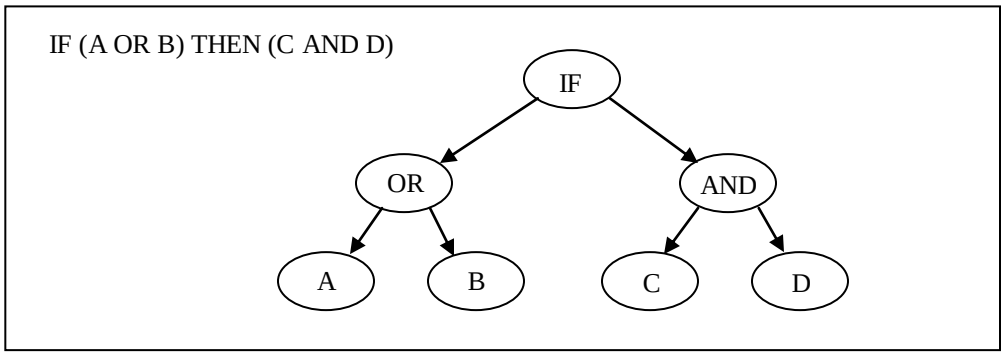

Figure 1: Binary parse tree representation (not CNF)

Most SAT solvers operate on Boolean statements that are in conjunctive normal form (CNF). The parse trees are transformed into CNF by application of the double negative law, the distributive law, and De Morgan's laws. After transformation the parse trees contain only AND, OR, NOT, and PRIMITIVE nodes (IF has been reduced). Further, NOT nodes have been pushed towards the leaves of the trees, existing only as parents to PRIMITIVE nodes. AND nodes have been pushed toward the root, such that no OR node has a child AND node.

3.2 SAT with Task Subsets

The information encoded in the object relation database defines all configurable products. Thus far we have derived a set of Boolean statements, each of which may evaluate to true or false depending on the true/false value of the related primitives. Let us call this total rule set Φ. It remains only to specify which tasks we wish to evaluate for satisfiability. If the entire set of tasks I_m assigned to the station can occur on a single product, then it follows that the primitives associated with those tasks can be set to true and the SAT can still be solved. In other words, in this case there would be some combination of true/false values of the option and platform variables that would result in all of these tasks being necessary for a single product. Using this concept, we present the following algorithm to determine the maximum time subset of tasks that may occur on a single product.

Algorithm *maxsubset*

1. Set $J = I_m$
2. Set $\widehat{\Phi} = \Phi$
3. Append rule $j = \text{true}$ to $\widehat{\Phi}$ for each $j \in J$
4. Solve SAT for $\widehat{\Phi}$. If SAT is true, stop. Else, continue

5. Save J to remember that it has been tested
6. Set J = the next smallest set of tasks from I_m
7. Got to Step 2.

End *maxsubset*

In this algorithm task sets are considered and tested in a sequence determined by the sum all task times included in the set. Initially all tasks at the station are considered, I_m. If that set is not feasible then the task with the smallest time is removed from the set, and the test is repeated. Step 5 remembers past tests to facilitate the search mechanism that must be performed in Step 6.

Conclusion

As the number of configuration options grows, production modeling methods that rely on the mixed-model paradigm increasingly struggle to enumerate the total number of unique models that might be produced. Further, specification of necessary input parameters such as demand may become infeasible for these large model sets. The options-mix paradigm offers hope for alternative modeling methods, as options sets can easily be enumerated and assigned e.g. demand. One of the primary challenges of the options-mix paradigm is that the information model no longer contains direct analogues of the production units. A horizontal line balancing problem is introduced in which examination of various production units is mandatory for evaluation of the needed objective function. A procedure for modeling options information is then presented that permits application of a SAT to the options-mix configuration data in order to deliver the necessary maximum time bound for the horizontal line balancing metric.

References

1. Pine, B.J., 1993. "Mass customization: the new frontier in business competition," Harvard: Harvard Business School Press.
2. Meyr, H., 2004. "Supply chain planning in the German automotive industry," OR Spectrum, 26, 447–470.
3. Roder, A. and Tibken, B., 2006, "A methodology for modeling inter-company supply chains and for evaluating a method of integrated product and process documentation," European Journal of Operational Research, 169, 1010–1029.
4. Scholl, A, 1999, "Balancing and sequencing assembly lines," 2nd ed. Heidelberg, Physica.
5. Shtub, A, and Dar-El, E., 1989, "A methodology for the selection of assembly systems," International Journal of Production Research 27, 175-186.
6. Mather, H, 1989, "Competitive manufacturing," Prentice Hall, Englewood Cliffs, NJ.
7. Boysen, N., Fliedner, M., and Scholl, A., 2009, "Production planning of mixed-model assembly lines: overview and extensions," Production Planning & Control: The Management of Operations, 20:5, 455-471.
8. Dolgui, A., Guschinsky, N., and Levin, G., 2006, "A special case of transfer lines balancing by graph approach," European Journal of Operational Research, 168, 732–746.
9. Burns, L., and Daganzo, C., 1987, "Assembly line job sequencing principles," International Journal of Production Research 25, 71-99.
10. Dobson, G., and Yano, C., 1994, "Cyclic scheduling to minimize inventory in a batch flow line," European Journal of Operational Research 75, 441-461.
11. Wee, T., and Magazine, M., 1982, "Assembly line balancing as generalized bin packing," Operations Research Letters 1/2, 56-59.
12. Thomopoulos, N., 1970, "Mixed model line balancing with smoothed station assignments," Management Science 16, 593-603.
13. Macaskill, J., 1972, "Production-line balances for mixed-model lines," Management Science 19, 423-434.
14. Domschke, W., Klein, R., and Scholl, A., 1996, "Antizipative Leistungabstimmung bei moderner Variantenfließfertigung," Zeitschrift für Betriebswirtschaft 66, 1465-1490.
15. Vilarinho, P., and Simaria, A., 2002, "A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations," International Journal of Production Research 40, 1405-1420.
16. Yano, C., and Bolat, A., 1989, "Survey, development, and application of algorithms for sequencing paced assembly lines," Journal of Manufacturing and Operations Management 2, 172-198.
17. Sumichrast, R., and Russell, R., 1990, "Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems," Journal of Operations Management 9, 371-390.
18. Sumichrast, R., Russell, R., and Taylor, B., 1992, "A comparative analysis of sequencing procedures for mixed-model assembly lines in a just-in-time production system," International Journal of Production Research 30, 199-214.
19. Bard, J., Dar-El, E., and Shtub, A., 1992, "An analytic framework for sequencing mixed model assembly lines," International Journal of Production Research 30, 35-48.
20. Merengo, C., Nava, F., and Pozetti, A., 1999, "Balancing and sequencing manual mixed-model assembly lines," International Journal of Production Research 37, 2835-2860.
21. Emde, S., Boysen, N., and Scholl, A., 2010, "Balancing mixed-model assembly lines: a computational evaluation of objectives to smoothen workload," International Journal of Production Research, 48:11, 3173-3191.