Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 5102616, 13 pages
http://dx.doi.org/10.1155/2016/5102616

Research Article

A Hybrid Programming Framework for Modeling and Solving Constraint Satisfaction and Optimization Problems

Paweł Sitek and Jarosław Wikarek

Department of Information Systems, Kielce University of Technology, 25-314 Kielce, Poland

Correspondence should be addressed to Paweł Sitek; sitek@tu.kielce.pl

Received 2 February 2016; Revised 25 May 2016; Accepted 21 June 2016

Academic Editor: Can Özturan

Copyright © 2016 P. Sitek and J. Wikarek. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper proposes a hybrid programming framework for modeling and solving of constraint satisfaction problems (CSPs) and constraint optimization problems (COPs). Two paradigms, CLP (constraint logic programming) and MP (mathematical programming), are integrated in the framework. The integration is supplemented with the original method of problem transformation, used in the framework as a presolving method. The transformation substantially reduces the feasible solution space. The framework automatically generates CSP and COP models based on current values of data instances, questions asked by a user, and set of predicates and facts of the problem being modeled, which altogether constitute a knowledge database for the given problem. This dynamic generation of dedicated models, based on the knowledge base, together with the parameters changing externally, for example, the user's questions, is the implementation of the autonomous search concept. The models are solved using the internal or external solvers integrated with the framework. The architecture of the framework as well as its implementation outline is also included in the paper. The effectiveness of the framework regarding the modeling and solution search is assessed through the illustrative examples relating to scheduling problems with additional constrained resources.

1. Introduction

Constraint satisfaction problems (CSPs) and/or constraint optimization problems (COPs) can involve the variables that take values over finite domains (integer, real, binary, etc.) and constraints of all types and characters [1]. By connecting variables, constraints affect the feasible variable domain ranges. Modeling and solving of those problems make up one of the major interest areas of various computer science communities, including operation research, mathematical programming, constraint programming, and artificial intelligence. Problems with constraints like CSP and COP are frequent in production, distribution, transportation, logistics, computer networks, software engineering, project management, planning and scheduling, and so forth. One of the features resulting from the users' changeable expectations is the need to solve the problem multiple times for variable data instances and parameters. Users express their expectations by asking all kinds of questions. On the one hand, the question is related to the possibility of realizing the task with certain resources at the defined time; on the other hand, it concerns optimal parameters of task realization, and it is about the optimal configuration of the system. Quite often, the questions include logical conditions (e.g., relating to mutual exclusion, dynamic connecting of resources). Because of the changeability of the questions, parameters, and data instances, the idea of autonomous search seems to be the most suitable for solving the problems with constraints. The autonomous search should have the ability to preferably modify and change its internal components when exposed to changing external parameters, requirements, and/or data instances [2].

The underlying motivation for this study was the idea of developing a programming and implementation platform, which would allow effective modeling and solving of CSPs and COPs and solving these problems in an automatic mode (using the autonomous search) despite changes in data instances, parameters, and questions asked by users. The idea was implemented as a hybrid programming framework. To build the framework, the authors used hybridization of various programming paradigms and their own original method of transformation. In addition, the authors proposed

a dynamic method for generating dedicated models, based on the knowledge database made up of facts, predicates, and questions asked by users.

2. Backgrounds, Methods, and Structures

Models for problems with constraints need environments that allow modeling and solving the constraints in an easy and effective way. Historically, operations research, in particular, mathematical programming, network programming, and dynamic programming, and so forth, has been used for this purpose. Numerous models (MIP-mixed integer programming, MILP-mixed integer linear programming, IP-integer programming, etc.), algorithms (branch and bound, symplex, branch and cost, etc.), and good practices have been developed to facilitate solving problems with constraints [3]. All these methods and models, however, have some limitations concerning the character of constraints (e.g., only linear constraints) or the character of variables (e.g., only real variables), and so forth. For different types of constraints (nonlinear, logical, etc.) and/or decision variables (integer, binary, etc.), they were either inapplicable or ineffective. For the approach to be most universal and suitable, a given problem must be looked at from the perspective of variables and connecting constraints with the domains of variables taken into account. Constraint logic programming (CLP) paradigm allows this approach. Constraint logic programming (CLP) is a form of constraint programming (CP) paradigm, where logic programming is extended to include CSP (constraint satisfaction problem). CLP programs are built from valid Prolog-based logic data structures. A program is a collection of predicates, and a predicate is a collection of clauses. The idea of a clause is to define that something is true. The simplest form of a clause is the fact. For example, the following two are facts: technology (product, machine, and execution_time) and vehicle (capacity, type, and cost). Syntactically, a fact is just a structure (or an atom) terminated by a full stop [4].

CSP is a triple $(X, \mathrm{Dom}, \mathrm{Cst})$ where $X = \{X_1, X_2, \ldots, X_m\}$ represents a set of m decision variables, $\mathrm{Dom} = \{\mathrm{Dom}^1, \mathrm{Dom}^2, \ldots, \mathrm{Dom}^m\}$ represents the set of associated domains (i.e., possible values for decision variables), and $\mathrm{Cst} = \{\mathrm{Cst}^1, \mathrm{Cst}^2, \ldots, \mathrm{Cst}^n\}$ represents a set of n constraints. Constraints fall into several types depending on the number of decision variables in a constraint (unary, binary, and n-ary). A unary constraint is a constraint on a single decision variable (e.g., $X \neq 6$, $Y < 6$). A binary constraint is a constraint over a pair of decision variables (e.g., $X > Y$, $X + Y < 8$). In general, a n-ary constraint has a scope of size n decision variables [1].

Each constraint Cst^i binds a set of decision variables and is used to restrict domains of these variables. Solving a CSP means finding the state/condition of a problem, in which the assignment of decision variables satisfies all constraints. The general algorithm for solving a CSP is shown in Figure 1. The algorithm consists of constraint propagation and variable distribution activated in the sequence. If this sequence does not provide a result, backtracking is used and sequence activation is repeated. The algorithm is very effective for solving the problems, in which aryness of constraints does not exceed 2. The CSP algorithm is often ineffective in the case of the problems in which constraints connect more than two decision variables and the optimization problems with constraints (COPs). The effectiveness of propagation is reduced significantly and the number of backtrackings increases. In extreme cases, the algorithm is able to neither find any feasible solution within the allowable time nor ascertain its absence.

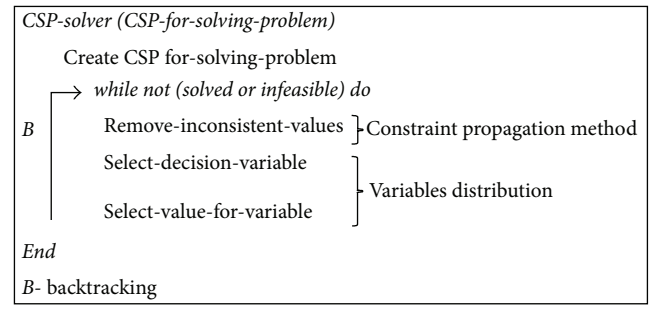


Figure 1: The general scheme of the algorithm to solve CSP.

In order to overcome this shortcoming, suggestions of integrating the CP/CLP paradigms with other paradigms occurred. Since the areas are similar, constraints and decision variables, the integration usually relates to the paradigm of mathematical programming [5–9].

Several scenarios of CP/CLP and MP integration have been reported in the literature [10]:

(i) Double modeling uses both CP and MP models and exchanges information while solving.

(ii) Search-inference duality views CP and MP methods as special cases of a search/inference duality.

(iii) Decomposition decomposes problems into a CP part and an MP part using a Benders scheme.

In the approach proposed in this paper, the scenario for the integration of both paradigms is supplemented with the authors' own method of problem transformation [11–13] and the idea of autonomous search [12]. All these components are connected and integrated into the programming hybrid framework (Section 3).

The method of problem transformation [11, 13] proposed by authors (Section 3.2) is briefly speaking, the transformation of the search space performed by removing all points and areas in which decision variables cannot occur, thus reducing the size of this space. Transformations are usually performed through changing the problem specification and adding the transformed model (with reduced and transformed decision variables and constraints). The transformation is conducted on the basis of facts, attributes, and constraints. The idea of autonomous search (Section 3.3) is implemented through the transformation method and automatic model generation. The models are generated dynamically using adequate CLP predicates, based on the current set of input facts (data instances), knowledge database (with facts and predicates describing a given problem), and question(s) asked by users. Both

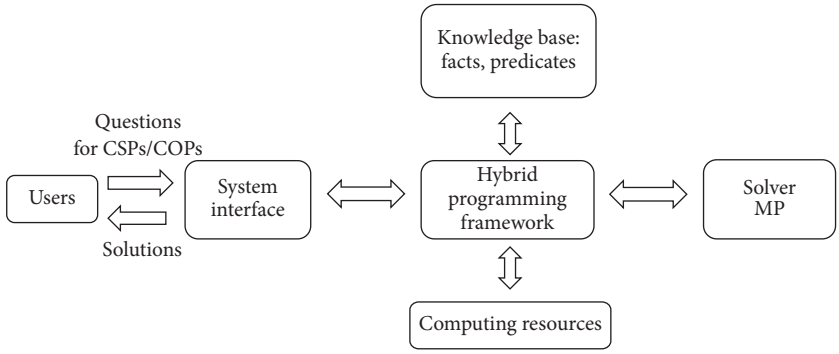


Figure 2: The concept of hybrid programming framework and its environment.

the input facts and the questions can change dynamically, which leads to new model generation/formation.

The main contribution of this study is the concept and implementation of the hybrid programming framework, which joins the ideas of (i) hybridization in the form of integration of MP and CLP, (ii) presolving in the form of transformation and constraint propagation, and (iii) autonomous search in the form of automatic generation of dedicated models to solve. Additionally, formal models of the scheduling problems for the illustrative examples before and after transformation are included.

3. The Concept of Hybrid Programming Framework Using Idea of an Autonomous Search

Based on the experience of hybridization and integration of CP/CLP/MP [5–8, 11, 13], the hybrid programing framework to modeling and solving CSPs and COPs has been proposed.

The main assumptions used in the concept and implementation of the hybrid programming framework were as follows:

(i) Integration of CLP and MP environments.

(ii) Introducing the framework presolving methods in the form of transformation and constraint propagation.

(iii) Knowledge base, which contains predicates for constraints, questions, methods, tools, and so forth and facts for data instances.

(iv) Implementation autonomous search in the form of automatic models generation for CSPs and COPs as the MP/MIP/MILP models based on knowledge base (constraints, questions, and data facts).

(v) The ability to solve MIP/MILP/MP models by internal and external solvers (LINGO [14] or SCIP [15] in this version of framework).

(vi) Replacing the variable distribution methods (Figure 1) through MP methods and algorithms (e.g., branch and bound, cutting plane, relaxation, etc.).

(vii) Implementation of the framework using the CLP environment (ECL^{i}PS^e system [16]).

3.1. Architecture of Hybrid Programming Framework. Figure 2 shows a context scheme of the framework. The framework communicates with the knowledge database, the user, and the external MP solver(s). The knowledge database, which de facto is a part of the framework, consists of a set of various types of predicates, including those most simple facts. The data instances of a given problem are saved as facts. Relationships between individual facts define the information structure of the problem. Predicates and facts may concern different problems modeled and solved using the framework and for this reason they are identified through the problem index (id_pro). The set of facts can be logically divided into two subsets: the subset of constant facts describing the problem and the subset changing the input facts. A user communicates with the framework by sending an inquiry/question in a suitable format and structure:

$$\text{Question}\ (\text{type},\ \text{parameters},\ \text{ID\_pro}). \tag{1}$$

The question determines what problem will be solved, with what parameters (i.e., which facts will be used), and defines the type of the question (which evaluation criterion will be used). An appropriate dedicated model will be generated depending on the formalization of this question, in particular, on its type.

Predicates can be logically divided into several groups. Particular groups of predicates (except for facts) that create the knowledge base are shown in Description of the Group Predicates.

A simplified functional scheme of the framework is shown in Figure 3; Algorithm 1 depicts the underlying/basic scenario of the framework operation, in the form of a pseudocode.

The user's question initiates the framework operation. The structure of the question in (1) defines the type of the question (general, wh-question, logical, etc.) and type of the problem and its detailed parameters (time, number of resources, size, etc.). Depending on the question, adequate information is extracted about the problem, data instances, problem size, and so forth.

Based on this information, the framework downloads suitable facts and predicates from the knowledge database. The facts are converted into lists. The reference model is built in CLP for a CSP or COP (variables and constraints).

```
While new question
    Determining the type of the problem (based of questions)
    Determining the parameters of the question
    Determining the size of the problem (based on facts)
    Initiate basic variables
    Load data about the problem from set of facts
    Basic constraints (load predicates)
    while additional conditions resulting from question
        Initiate additional variables
        Additional constraints (load predicates)
     Starting constraint propagation
     if transformation of the problem then
        Transformation
        Starting constraint propagation
     Determining the type of the MP Solver
     Generation of the MP model in the Solver format or MPS
     Start Solver
```

Algorithm 1: The basic scenario of the framework operation.

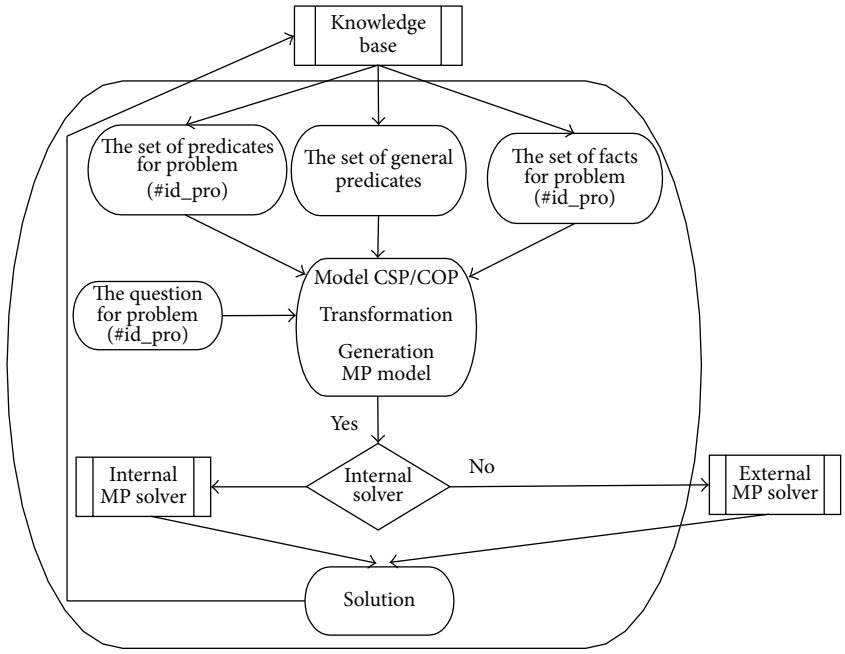


Figure 3: The functional diagram of hybrid programming framework.

The model can be supplemented with additional constraints, including logic constraints if necessary. In the next step, the model is subjected to presolving. Two presolving methods are used in the framework, usually alternately: constraint propagation and transformation. After presolving, the model is the basis for generating the final implementation model in the form of an MP model (usually MILP). The MP model is then solved using an MP solver.

3.2. Transformation. Transformation has been studied by the authors [11, 13]. It consists in changing the specification of a problem to eliminate unacceptable/nonfeasible points from the solution space prior to solving the problem. As a result, the number of decision variables is reduced and aggregated and the constraints are simplified and also the numbers of constraints is reduced, which leads to a smaller search space and, hence, shorter search time and the possibility to solve problems of larger size within allowable time limits. Facts and problem constraints are used in the process of transformation. Transformation and constraint propagation are presolving methods applied in the framework. In real practical problems, the transformation may involve the removal of unacceptable transportation routes in SCM problems [13] and the change of the problem specification from operational to resource type in task group scheduling [17], and so forth. For the illustration example (Section 4),

the transformation relies on the change problem representation through the appropriate aggregation of indices. For allowable values of indices of machines and products, an aggregated implementation index is created, the values of which are determined based on feasible values of base indices ((2a), (2b)). Details of the transformation in terms of indices, decision variables, constraints, and facts for the illustration example are presented in Section 4 and Appendices D.1 and D.2.

3.3. Autonomous Search. Autonomous search, as used in the framework, is the narrowing of the search space through the implementation of presolving methods and automatic generation of dedicated implementation models. Both the presolving methods (constraint propagation and transformation) and model generation are based on current data instances. Users' questions are also taken into account while constructing the model. The questions may be related only to some of the aspects and constraints of the problem. Based on the current data, instances and users' questions ensure that the automatically generated models are dedicated and fit the specific situation. Such dedicated models have fewer decision variables and constraints.

This shortens the search time (the search space is considerably reduced relative to that in universal models). A change of the question and/or data instance results in a new model adjusted to new parameters.

4. Illustrative Example and Computational Experiments

Practical use of framework for modeling and solving problems will be presented for illustrative example. As an illustrative example was selected job-shop scheduling problem with additional resources [18, 19]. Problems of this type can be found in manufacturing, logistics, computer networks, software engineering, and so forth.

Formally, the illustrative example is an extension variant of job-shop scheduling problem and can be defined as follows. A set of LI jobs/orders/products $I = \{I_1, I_2, \ldots, I_{LI}\}$ are given which require, for their processing, a set of LM machines $M = \{M_1, M_2, \ldots, M_{LM}\}$ and a set of LR additional resources $R = \{R_1, R_2, \ldots, R_{LR}\}$. Each additional resource R_r has a specified limit ko_r (number of units of the resource R_k). Each job/order I_l is a sequence of k operations. The kth operation of job/order I_i has to be executed by a specific machine $M_k \in M$ for time units ($\mathrm{tr}_{i,k}$ is integer). Generally, in job-shop problems, a feasible schedule is such that (a) at any time each machine can execute at most one operation, (b) the operations of the same job/order are totally ordered, and (c) no preemption is allowed. Moreover, in our example, any time, each operation can be assisted by additional resources R_r where $d_{i,m,r}$ determines how much additional resources R_r are used to execute job/order I_i on machine $M_k \in M$. Additional resources can be operators, tools, memory, transportation units, and so forth while basic resources are the machines/processors/workstations. A formal model of a scheduling problem for illustrative example, containing constraints, decision variables, and parameters, is shown in Appendix D.1. The collection of facts together with their structure for this model is included in Figure 4 (as the lowest layer of the information structure).

Both the proposed model and the structure of facts constitute a significant extension of the classical job-shop scheduling problem (JSSP) [18, 19]. Firstly, they allow accounting for additional resources R, as described above. Secondly, the structure of constraints of the model, decision variables, and facts is universal and can describe not only job-shop scheduling problems but also those in other environments including flow-shop, open-shop, project, multiproject, and so forth.

Modeling starts with loading the set of facts for illustrative example to the knowledge base. Then, the facts are converted into lists using a general predicate (P1). In the next step, the set of predicates (P2) is created. This set implements the basic and additional/logic constraints for illustrative example. Then, the built or expanded set of predicates (P3) implements various types of questions (e.g., as in the exemplified questions asked to the illustrative example) for illustrative example. In the next step, predicates to transform modeled problem (P4) are taken from the knowledge base. Transformation for illustrative example involves the aggregation of the relevant facts (indexes of these facts) and building a list of only the feasible combinations of facts. The principle of the transformation of the facts for illustrative example is shown in Figure 5.

Exemplified questions asked to the illustrative example are as follows

(Q1) What is the min $C_{\max}$ (makespan)?

(Q2$_A$) What is the min $C_{\max}$ if the set of additional resources is $\mathrm{ko}_1 = \mathrm{ko}_2 = \mathrm{ko}_3 = \mathrm{ko}_4 = 2$?

(Q2$_B$) What is the min $C_{\max}$ if the set of additional resources is $\mathrm{ko}_1 = \mathrm{ko}_2 = \mathrm{ko}_3 = \mathrm{ko}_4 = 3$?

(Q3) What is the minimum set of resources R_1 at $C'_{\max}$?

(Q4) Is it possible to schedule orders in $C'_{\max}$ and what are the sets of resources R_1, R_2, R_3, R_4?

(Q5) Is it possible to schedule orders in $C'_{\max}$ if resources R_1 and R_4 cannot be used simultaneously?

(Q6) Is it possible to schedule orders in $C'_{\max}$ if machines M_7 and M_9 cannot be used simultaneously?

(Q7) What is the min $C_{\max}$ if resources R_1 and R_4 cannot be used simultaneously?

(Q8$_A$) What is the min $C_{\max}$ if machines M_1 and M_2 cannot be used simultaneously?

(Q8$_B$) What is the min $C_{\max}$ if machines M_7 and M_9 cannot be used simultaneously?

The resulting list L_index for facts from Appendix B.1 are presented in the Appendix B.2. Indices (the dimensions of the problem), decision variables, and constraints of the model are also subject to transformation. The transformation of indices (2a) stems from the fact that not every product i has to be manufactured on every machine m. Thus, if product i is made on a specific machine m, the existing values of index pair

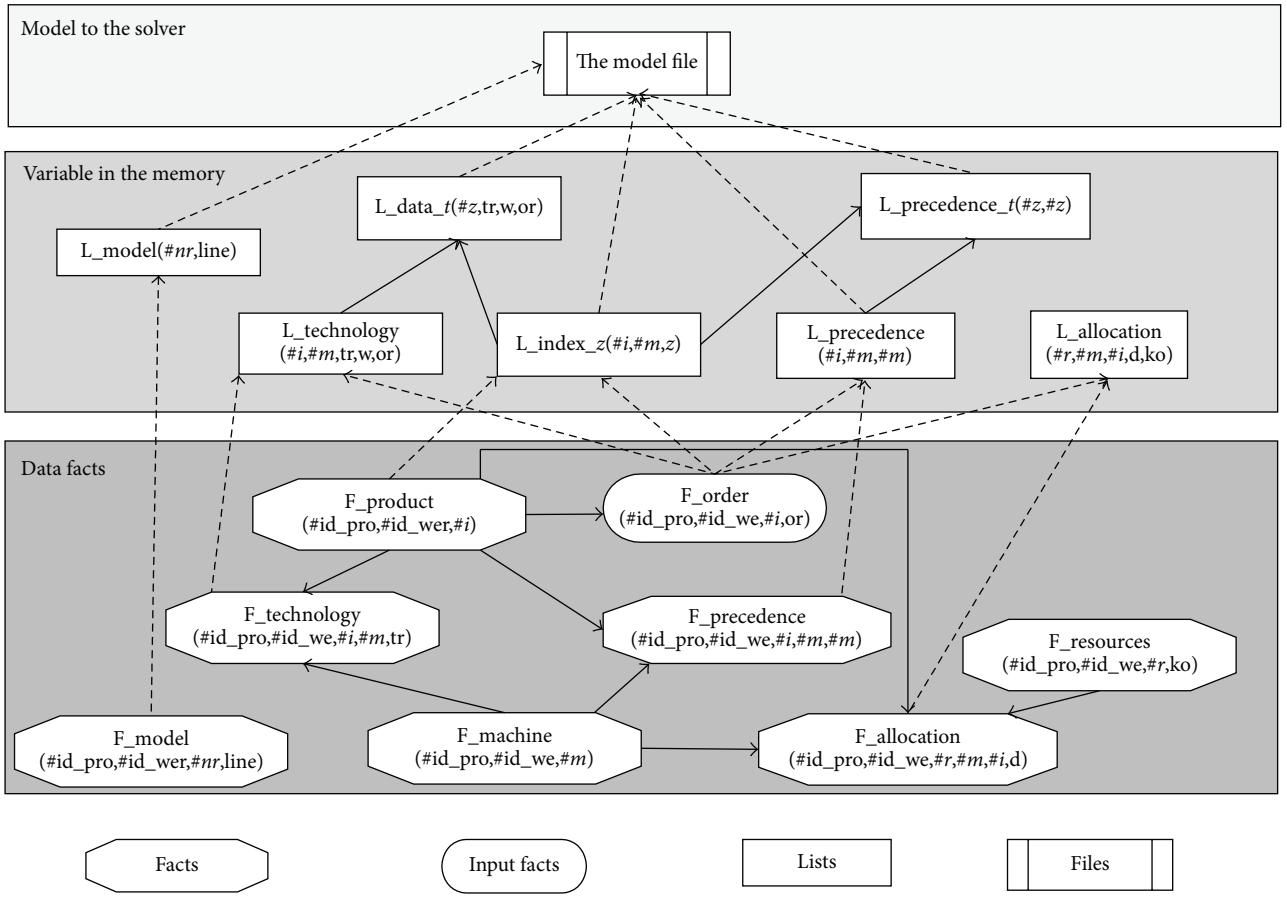


Figure 4: The information structure for illustrative example implemented in hybrid programming framework.

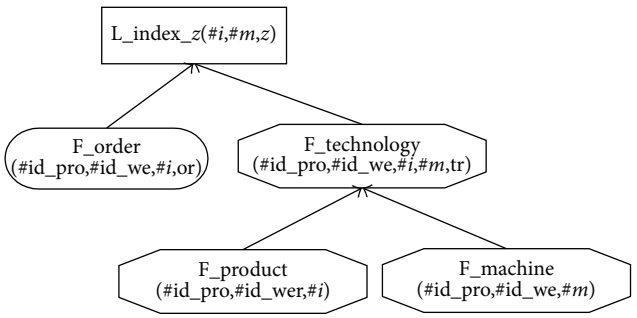


Figure 5: The principle of the transformation of the facts for illustrative example.

(i, m) are replaced with the values of aggregated/transformed index z. In the next step, the decision variables are subjected to transformation as a result of the aggregation of some of their indexes (2b). The set of all decision variables of the model before and after transformation are presented in Appendices D.1 and D.2, respectively. In the final stage, the constraints of the model transformed through the change (reduction) of summations and "for" phrase for the individual constraints. The constraints for model before and after the transformation are included in Appendices D.1 and D.2, respectively:

$$(i, m) \longrightarrow (z) \tag{2a}$$

$$X_{i,m,r,t} \longrightarrow X_{z,r,t} \mid \mathrm{Kp}_{i,m} \longrightarrow \mathrm{Kp}_{z} \mid Y_{i,m,r,t} \longrightarrow Y_{z,r,t}. \tag{2b}$$

The final step is the generation of a dedicated MILP model (files in the appropriate solver format or Mathematical Programming System (MPS)) using universal set of predicates for automatic generation (P5). Schematic structure of information for illustrative example in the form of facts, lists, and files is shown in Figure 4 and the description is shown in Table 3.

The scenario of computational experiments was as follows. For any questions from the exemplified questions asked to the illustrative example and a given set of data instances (Appendix B.1), the generation of dedicated MILP models have been made using a hybrid programming framework. After that, the automatically generated models were solved using the external MP solvers like "LINGO" [14] or "SCIP" [15]. Choosing solver "SCIP" was due to its high efficiency and the use of the built-in powerful presolving methods [15]. As for the efficiency and effectiveness in the area of MP and CP, SCIP is the best option of all noncommercial solvers [15].

Table 1: Results for asked questions to the illustrative example (framework implementation).

Questions	V	C	Parameters	Answer	T_1	T_2
Q1	3070	16377	—	$\min C_{\max} = 22$	7	6
$Q2_A$	3070	16377	$ko_1 = ko_2 = ko_3 = ko_4 = 2$	$\min C_{\max} = 26$	316	146
$Q2_B$	3070	16377	$ko_1 = ko_2 = ko_3 = ko_4 = 3$	$\min C_{\max} = 22$	60	28
Q3	3070	16377	$C'_{\max} = 24$	$ko_{1\min} = 2$,	2	2
Q4	3270	16678	$C'_{\max} = 24$	Yes $ko_1 = 3, ko_2 = 4, ko_3 = 4, ko_4 = 2$	2	2
			$C'_{\max} = 20$	No	1	1
			$C'_{\max} = 23$	Yes $ko_1 = 4, ko_2 = 4, ko_3 = 3, ko_4 = 2$	2	1
Q5	3266	16623	$C'_{\max} = 24$	NO	2	2
			$C'_{\max} = 30$	YES	12	6
Q6	3560	16917	$C'_{\max} = 24$	NO	3	2
			$C'_{\max} = 26$	YES	6	2
Q7	3266	16623		$\min C_{\max} = 27$	16	8
$Q8_A$	3560	16917		$\min C_{\max} = 22$	6	5
$Q8_B$	3560	16917		$\min C_{\max} = 26$	4	3

Table 2: Results for asked questions to the illustrative example (MP implementation).

Questions	V	C	Parameters	Answer	T_1	T_2
Q1	70604	75343	—	$C_{\max} = 22$	128	67
$Q2_A$	70604	75343	$ko_1 = ko_2 = ko_3 = ko_4 = 2$	$C_{\max} = 26$	3756	1546
$Q2_B$	70604	75343	$ko_1 = ko_2 = ko_3 = ko_4 = 3$	$C_{\max} = 22$	546	234

$\min C_{\max}$: optimal makespan.
$C'_{\max}$: given makespan.
V: the number of decision variables.
C: the number of constraints.
T_1: time of finding solution (in seconds) in LINGO.
T_2: time of finding solution (in seconds) in SCIP.

CPLEX and Gurobi are certainly faster in solving the same benchmarks, but being commercial solvers, they mean higher costs (licenses, etc.).

Obtained results are shown in Table 1. The scope of the exemplified questions asked to the illustrative example shows the flexibility and capabilities of a hybrid programming framework. These are general questions (Q4, Q5, and Q6) and specified questions (Q1, $Q2_A$, $Q2_B$, and Q3), which require both optimal and feasible solutions. In addition, questions can be logical (Q7, Q8), whose modeling directly in an MP environment is not obvious and simple. To determine the effectiveness of the proposed framework for questions Q1, $Q2_A$, and $Q2_B$ (for the most demanding computing), models were generated using the framework (Table 1) and modeled using only the classical mathematical programming (Table 2). Then, both groups of models were solved using LINGO and SCIP solvers. The answers to these questions using the framework are obtained 10 to 20 times faster than using just the pure MP solvers. In each case, the use of "SCIP" solver accelerated calculations twofold in comparison with the "LINGO" solver, with the "LINGO" solver (see columns T_1 and T_2 of Tables 1 and 2).

The models generated using the framework, respectively, have 20 times smaller number of decision variables and 3 times smaller number of constraints in relation to the models created only in MP environments.

The model file for Q1 questions in a format compatible with the "LINGO" is shown in Appendix C.

The file was generated by group predicates P5 on the basis of the information structure (Figure 5).

5. Conclusions

The proposed hybrid framework can be used in two modes. Firstly, it can be a platform for the end user to solve the COP and the CSP, which are generated and solved on the basis of existing knowledge base, but also the questions asked by the user. Secondly, it is a programming framework for modeling and solving the COP and the CSP. In this case, the user must know the environment CLP. For each new problem, users supplement the knowledge base of relevant predicates in P2 and P3 sets (see Description of the Group Predicates).

The concept of hybrid programming framework that combines (a) two programming paradigms (CLP/MP); (b) the presolving methods (transformation and constraint propagation); (c) autonomous search; and (d) the automatic generation of dedicated implementation models is the flexible and efficiency solution. Flexibility and easiness of modeling problems are caused by CLP-based approach which by nature is declarative. Efficiency is the result of applying the presolving methods, dedicated implementation models, and mathematical programming for solving. The idea of autonomous

Table 3: Description of the facts, lists, and parameters for the illustrative example.

Facts	Keys	Parameters
	Facts about structure of the problem	
F_product (#id_pro, #id_wer, #*i*)	#id_pro: problem ID #id_wer: version of the data instances #*i*: product ID	
F_machine (#id_pro, #id_wer, #*m*)	#id_pro: problem ID #id_wer: version of the data instances #*m*: machine ID	
F_technology (#id_pro, #id_wer, #*i*, #*m*, parameters)	#id_pro: problem ID #id_wer: version of the data instances #*i*: product ID #*m*: machine ID	$tr_{i,m}$: the execution time of the product *i* on the machine *m*
F_precedence (#id_pro, #id_we, #*i*, #*m*, #*m*)	#id_pro: problem ID #id_wer: version of the data instances #*i*: product ID #*m*: machine ID	
F_resources (#id_pro, #id_we, #*r*, parameters)	#id_pro: problem ID #id_wer: version of the data instances #*r*: additional resource ID	ko_r: the total number of additional resources *r*
F_allocation (#id_pro, #id_we, #*r*, #*m*, #*i*, parameters)	#id_pro: problem ID #id_wer: version of the data instances #*r*: additional resource ID #*m*: machine ID #*i*: product ID	$d_{i,m,r}$: the number of additional resources *r* needed to execute a product *i* on the machine *m*
F_model (#id_pro, #id_wer, #*nr*, parameters)	#id_pro: problem ID #id_wer: version of the data instances #*nr*: model ID	$line_{nr}$: line of code for model ID
F_order (#id_pro, #id_we, #*i*, parameters)	#id_pro: problem ID #id_wer: version of the data instances #*i*: product ID	or_i: the size of the order for product *i*
	Lists	
L_precedence (#*i*, #*m*, #*m*)	#*m*: machine ID #*i*: product ID	
L_technology (#*i*, #*m*, parameters)	#*i*: product ID #*m*: machine ID	$tr_{i,m}$: the execution time of the product *i* on the machine *m*; $w_{i,m}$: if the product *i* is executed on the machine *m*, $w_{i,m} = 1$; otherwise, $w_{i,m} = 0$; or_i: the size of the order for product *i*
L_allocation (#*r*, #*m*, #*i*, parameters)	#*r*: additional resource ID #*m*: machine ID #*i*: product ID	$d_{i,m,r}$: the number of additional resources *r* needed to execute a product *i* on the machine *m*; ko_r: the total number of additional resources *r*
L_model (#*nr*, parameters)	#*nr*: model ID	$line_{nr}$: line of code for model ID
L_index_*z* (#*i*, #*m*, parameters)	#*i*: product ID #*m*: machine ID	*z*: the new index after transformation
L_data_*t* (#*z*, parameters)	#*z*: the new index after transformation (combination of #*i* and #*m*)	tr_z: the execution time of the product *i* on the machine *m*; w_z: if the product *i* is executed on the machine, $w_z = 1$; otherwise, $w_z = 0$; or_z: the size of the order for product *i*
L_precedence_*t* (#*z*, #*z*)	#*z*: the new index after transformation (combination of #*i* and #*m*)	

search is implemented mainly through the mechanism of automatic generation of implementation models based on current data instances and the requirements of users (in the form of frequently asked questions) which means that models are better suited to current requirements and conditions and their solution requires less space search. The knowledge base of the framework, which is built from predicates and facts, provides scalability because knowledge base can be updated with new facts relating to existing models, predicates, and facts of new models and the facts resulting in answers to user questions and so on.

Further research will focus on two directions/areas. The first is to use a framework for modeling and solving other problems in the area of widely understood computer science. The second is the integration framework with other paradigms such as fuzzy logic and concurrent programming.

A new remotely accessible (e.g., in the cloud [20]) version of the framework is going to be developed. For licensing reasons, LINGO solver will be replaced by SCIP in this version.

Appendix

A. Summary Facts, Lists, and Parameters for Illustrative Example

See Table 3.

B. Data Instances for Illustrative Example

B.1. The Sets of Facts for Illustrative Example. See Algorithm 2.

B.2. The List of New Indices after Transformation. See Algorithm 3.

C. MILP Model Automatically Generated by Framework (Set of Predicates P5) in LINGO Format

See Algorithm 4.

D. Formal Models for Illustrative Example

D.1. Formal/Mathematical Model for Illustrative Example

Indices

m: machine/processor/workstation $m = 1, \ldots, LM$,

i: product/service type $i = 1, \ldots, LI$,

r: additional resource (employees, tools, transport units, etc.) $r = 1, \ldots, LR$,

t: period $t = 1, \ldots, LT$.

Parameters

$\mathrm{tr}_{i,m}$: the time required to make a product i on the machine m,

$w_{i,m}$: if the product i is made using a machine m, then $w_{i,m} = 1$; otherwise, $w_{i,m} = 0$,

ko_r: the number of additional resource types r,

$d_{i,m,r}$: if the additional resource r is used to make the product i on the machine m, then $d_{i,m,r}$ determines the number of additional resources r necessary for this execution; otherwise, $d_{i,m,r} = 0$,

$d1_{i,m,r}$: if the additional resource r is used to make the product i on the machine m, then $d1_{i,m,r} = 1$; otherwise, $d1_{i,m,r} = 0$,

$do_{i,m1,m2}$: if the operation of the product i on the machine $m1$ to be executed before the operation on the machine $m2$, then $do_{i,m1,m2} = 1$; otherwise, $do_{i,m1,m2} = 0$.

Inputs

or_i: demand/order for product i.

Auxiliary Parameters

op_t: coefficient for conversion number of periods t for the variable $\mathrm{op}_t = t$.

Decision Variables

$X_{i,m,r,t}$: if the additional resource r in period t is used to make the product i on the machine m, then $X_{i,m,r,t} = 1$; otherwise, $X_{i,m,r,t} = 0$,

$\mathrm{Kp}_{i,m}$: the number of last periods in which the product i is made on the machine m,

$Y_{i,m,r,t}$: if the period t is the latest in which the additional resource r is used to make the product i on the machine m, then $Y_{i,m,r,t} = 1$; otherwise, $Y_{i,m,r,t} = 0$,

$C_{\max}$: Makespan.

Constraints

(1) $\mathrm{Kp}_{i,m} \le C_{\max}\ \forall i = 1, \ldots, LI, m = 1, \ldots, LM$.

Determination of the makespan.

(2) $\sum_{t=1}^{LT} d1_{i,m,r} \cdot w_{i,m} \cdot X_{i,m,r,t} = \mathrm{tr}_{i,m} \cdot \mathrm{or}_i\ \forall i = 1, \ldots, LI, m = 1, \ldots, LM, r = 1, \ldots, LR : \mathrm{tr}_{i,m} > 0$.

The allocation of resources r to the machine during product realization.

(3) $\sum_{i=1}^{LI} \sum_{r=1}^{LR} d1_{i,m,r} X_{i,m,r,t} \le 1\ \forall m = 1, \ldots, LM, t = 1, \ldots, LT$.

The allocation of at most one product to the machine in a given period of time.

(4) $\sum_{i=1}^{LI} \sum_{m=1}^{LM} d_{i,m,r} \cdot X_{i,m,r,t} \le \mathrm{ko}_r\ \forall r = 1, \ldots, LR, t = 1, \ldots, LT$.

The limited availability of resources (capacity constraints).

(5) $X_{i,m,r,t-1} - X_{i,m,r,t} \le Y_{i,m,r,t-1}\ \forall i = 1, \ldots, LI, m = 1, \ldots, LM, r = 1, \ldots, LM, t = 2, \ldots, LT$.

$\sum_{t=1}^{LT} Y_{i,m,r,t} \le 1\ \forall i = 1, \ldots, LI, m = 1, \ldots, LM, r = 1, \ldots, LM$.

Operations cannot be interrupted.

```
%F_machine(#M).
F_machine (M1). F_machine(M2). F_machine(M3). F_machine(M4). F_machine(M5).
F_machine(M6). F_machine(M7). F_machine(M8). F_machine(M9). F_machine(M10).
F_machine(M11). F_machine(M12).
%F_product(#I).
F_product(A). F_product(B). F_product(C). F_product(D). F_product(E).
F_product(F). F_product(G). F_product(H). F_product(I). F_product(J).
F_product(K). F_product(L). F_product(M). F_product(N). F_product(O).
%technology(#I,#M,tr).
F_technology(A,M1,1). F_technology(A,M2,2). F_technology(A,M3,2).
F_technology(A,M10,1). F_technology(B,M1,1). F_technology(B,M5,2).
F_technology(B,M8,1). F_technology(C,M4,2). F_technology(C,M9,4).
F_technology(D,M5,2). F_technology(D,M6,2). F_technology(D,M7,5).
F_technology(D,M8,2). F_technology(E,M1,2). F_technology(E,M2,1).
F_technology(E,M3,2). F_technology(E,M4,2). F_technology(F,M5,2).
F_technology(F,M6,2). F_technology(G,M3,1). F_technology(G,M5,2).
F_technology(G,M8,2). F_technology(H,M8,1). F_technology(H,M9,1).
F_technology(H,M10,1). F_technology(I,M6,1). F_technology(I,M7,1).
F_technology(I,M8,1). F_technology(J,M4,1). F_technology(J,M5,1).
F_technology(J,M6,1). F_technology(K,M10,1). F_technology(K,M11,1).
F_technology(K,M12,1). F_technology(L,M1,2). F_technology(L,M11,2).
F_technology(L,M12,2). F_technology(M,M9,1). F_technology(M,M10,1).
F_technology(M,M11,2). F_technology(N,M1,2). F_technology(N,M12,2).
F_technology(O,M2,2). F_technology(O,M11,2).
%resources (#R,ko).
F_resources(R1,8). F_resources(R2,8).
F_resources(R3,8).F_resources (R4,8).
%allocation(#R,#M,#I,d)
F_allocation(R1,A,M1,1). F_allocation(R2,A,M2,1). F_allocation(R2,A,M3,2).
F_allocation(R3,A,M10,1). F_allocation(R1,B,M1,2). F_allocation(R3,B,M5,2).
F_allocation(R1,B,M8,1). F_allocation(R2,C,M4,2). F_allocation(R2,C,M9,2).
F_allocation(R3,D,M5,2). F_allocation(R1,D,M6,1). F_allocation(R3,D,M7,1).
F_allocation(R4,D,M8,2). F_allocation(R1,E,M1,1). F_allocation(R1,E,M2,1).
F_allocation(R3,E,M3,2). F_allocation(R3,E,M3,1). F_allocation(R3,F,M5,1).
F_allocation(R1,F,M6,2). F_allocation(R4,G,M3,1). F_allocation(R3,G,M5,2).
F_allocation(R1,G,M8,2). F_allocation(R2,H,M8,1). F_allocation(R1,H,M9,2).
F_allocation(R3,H,M10,2). F_allocation(R3,I,M6,1). F_allocation(R1,I,M7,1).
F_allocation(R2,I,M8,1). F_allocation(R4,J,M4,1). F_allocation(R3,J,M5,1).
F_allocation(R3,J,M6,1). F_allocation(R4,K,M10,1). F_allocation(R3,K,M11,1).
F_allocation(R1,K,M12,2). F_allocation(R2,L,M1,1). F_allocation(R1,L,M11,2).
F_allocation(R3,L,M12,2). F_allocation(R1,M,M9,1). F_allocation(R1,M,M10,1).
F_allocation(R2,M,M11,2). F_allocation(R3,N,M1,1). F_allocation(R2,N,M12,2).
F_allocation(R3,O,M2,2). F_allocation(R3,O,M11,2).
%precedence(#I,#M,#M).
F_precedence(A,M1,M2). F_precedence(A,M2,M3). F_precedence(A,M3,M10).
F_precedence(B,M1,M5). F_precedence(B,M5,M8). F_precedence(C,M4,M9).
F_precedence(D,M5,M6). F_precedence(D,M6,M7). F_precedence(D,M7,M8).
F_precedence(E,M1,M2). F_precedence(E,M2,M3). F_precedence(E,M3,M4).
F_precedence(F,M5,M6). F_precedence(G,M3,M5). F_precedence(G,M5,M8).
F_precedence(H,M8,M9). F_precedence(H,M9,M10). F_precedence(I,M6,M7).
F_precedence(I,M7,M8). F_precedence(J,M4,M5). F_precedence(J,M5,M6).
F_precedence(K,M10,M11). F_precedence(K,M11,M12). F_precedence(L,M1,M11).
F_precedence(L,M11,M12). F_precedence(M,M9,M10). F_precedence(m,M10,M11).
F_precedence(N,M1,M12). F_precedence(O,M2,M11).
orders(#I,or).
F_order(A,1). F_order(B,1). F_order(C,2). F_order(D,2). F_order(E,2).
F_order(F,1). F_order(G,1). F_order(F,1). F_order(G,1). F_order(F,1).
```

Algorithm 2: Instances of facts for illustrative example.

```
L_index_z=[[A,M1,1],   [A,M2,2],  [A,M3,3],  [A,M10,4], [B,M1,5],  [B,M5,6],
           [B,M8,7],   [C,M4,8],  [C,M9,9],  [D,M5,10], [D,M6,11], [D,M7,12],
           [D,M8,13],  [E,M1,14], [E,M2,15], [E,M3,16], [E,M4,17], [F,M5,18],
           [F,M6,19],  [G,M3,20], [G,M5,21], [G,M8,22], [H,M8,23], [H,M9,24],
           [H,M10,25], [I,M6,26], [I,M7,27], [I,M8,28], [J,M4,29], [J,M5,30],
           [J,M6,31]
         ]
```

Algorithm 3: The list of new indices after transformation.

(6) $\mathrm{Kp}_{i,m} = \sum_{t=1}^{LT}(\mathrm{op}_t \cdot Y_{i,m,r,t})\ \forall i = 1,\ldots,LM, m = 1,\ldots, LM, r = 1,\ldots,LR : \mathrm{tr}_{i,m} > 0, d1_{i,m,r} = 1.$

Determination of the time of the end product realization on the machine.

(7) $\mathrm{Kp}_{i,m2} - \mathrm{or}_i \cdot \mathrm{tr}_{i,m2} \geq \mathrm{Kp}_{i,m1}\ \forall i = 1,\ldots,LI, m1, m2 = 1,\ldots,LM : do_{i,m1,m2} = 1.$

The sequence of operations (precedence constraints).

(8) $X_{i,m,r,t} \in \{0,1\}\ \forall i = 1,\ldots,LI, m = 1,\ldots,LM, r = 1,\ldots,LR, t = 1,\ldots,LT.$

$Y_{i,m,r,t} \in \{0,1\}\ \forall i = 1,\ldots,LI, m = 1,\ldots,LM, r = 1,\ldots,LR, t = 1,\ldots,LT.$

$\mathrm{Kp}_{i,m} \in C\ \forall i = 1,\ldots,LI, m = 1,\ldots,LM.$

Binary values and integer values.

D.2. Formal/Mathematical Model for Illustrative Example after Transformation

Indices

m: machine/processor/workstation $m = 1,\ldots,LM$,

r: additional resource $r = 1,\ldots,LR$,

t: period $t = 1,\ldots,LT$,

z: implementation $z = 1,\ldots,Z$ and index after transformation (combined indices i, m).

Parameters

tr_z: the time required to make implementation z,

ko_r: the number of additional resources r,

$d_{z,r}$: if the additional resource r is used to make implementation z, then $d_{z,r}$ determines the number of additional resources r necessary for this implementation; otherwise, $d_{z,r} = 0$,

$d1_{z,r}$: if the additional resource r is used to make implementation z, then $d1_{z,r} = 1$; otherwise, $d1_{z,r} = 0$,

$do_{z1,z2}$: if the implementation $z1$ to be executed before the implementation $z2$, then $do_{z1,z2} = 1$; otherwise, $do_{z1,z2} = 0$,

$\mathrm{wyk}_{z,m}$: if the implementation z is made using a machine m, then $\mathrm{wyk}_{z,m} = 1$; otherwise, $\mathrm{wyk}_{z,m} = 0$.

Inputs

or_z: demand/order for implementation z.

Auxiliary Parameters

op_t: coefficients for conversion numbers of periods t for the variables $\mathrm{op}_t = t$.

Decision Variables

$X_{z,r,t}$: if the additional resource r in period t is used in implementation z, then $X_{z,r,t} = 1$; otherwise, $X_{z,r,t} = 0$,

Kp_z: the number of last periods in which the implementation z is made,

$Y_{z,r,t}$: if the period t is the latest in which the additional resource r is used in implementation z, then $Y_{z,r,t} = 1$; otherwise, $Y_{z,r,t} = 0$,

$C_{\max}$: Makespan.

Constraints

(1) $\mathrm{Kp}_z \leq C_{\max}\ \forall z = 1,\ldots,LZ.$

Determination of the makespan.

(2) $\sum_{t=1}^{LT} d1_{z,r} \cdot X_{z,r,t} = \mathrm{tr}_z \cdot \mathrm{or}_z\ \forall z = 1,\ldots,LZ, r = 1,\ldots,LR, : \mathrm{tr}_z > 0,\ d1_{z,r} = 1.$

The allocation of resources r to the machine during product realization.

(3) $\sum_{z=1}^{LZ}\sum_{r=1}^{LR} \mathrm{wyk}_{z,m} X_{z,r,t} \leq 1\ \forall m = 1,\ldots,LM, t = 1,\ldots,LT.$

The allocation of at most one product to the machine in a given period of time.

(4) $\sum_{z=1}^{LZ} d_{z,r} \cdot X_{z,r,t} \leq \mathrm{ko}_r\ \forall r = 1,\ldots,LR, t = 1,\ldots,LT.$

The limited availability of resources (capacity constraints).

(5) $X_{z,r,t-1} - X_{z,r,t} \leq Y_{z,r,t-1}\ \forall z = 1,\ldots,LZ, r = 1,\ldots, LM, t = 2,\ldots,LT.$

$\sum_{t=1}^{LT} Y_{z,r,t} \leq 1\ \forall z = 1,\ldots,LZ, r = 1,\ldots,LM.$

Operations cannot be interrupted.

(6) $\mathrm{Kp}_z = \sum_{t=1}^{LT}(\mathrm{op}_t \cdot Y_{z,r,t})\ \forall z = 1,\ldots,LZ, r = 1,\ldots,LR : \mathrm{tr}_z > 0, d1_{z,r} = 1.$

Determination of the time of the end product realization on the machine.

(7) $\mathrm{Kp}_{z2} - \mathrm{or}_{z2} \cdot \mathrm{tr}_{z2} \geq \mathrm{Kp}_{z1}\ \forall z1, z2 = 1,\ldots,LZ : do_{z1,z2} = 1.$

The sequence of operations (precedence constraints).

```
Model:
 Sets:
  machines     /1..@file(t_02_sizes.ldt)/;
  resources    /1..@file(t_02_sizes.ldt)/:ko;
  periods      /1..@file(t_02_sizes.ldt)/:op;
  transformed /1..@file(t_02_sizes.ldt)/:KP,or,tr;
  technology (transformed,resources,periods):X,Y;
  auxiliary_1 (transformed,resources):d,d1;
  auxiliary_2 (transformed,machines):wyk;
  auxiliary_3 (transformed,transformed):do;
 EndSets
 Data:
  or   =@file(t_03_data.ldt);  tr  =@file(t_03_data.ldt);
  wyk  =@file(t_03_data.ldt);  ko  =@file(t_03_data.ldt);
  d    =@file(t_03_data1.ldt); d1  =@file(t_03_data1.ldt);
  do   =@file(t_03_data1.ldt);
 EndData
 SUBMODEL F_objective1:
  Min=Cmax;
 ENDSUBMODEL
 SUBMODEL Constraints:
  @for(transformed(z): KP(z)<=Cmax);
  @for(transformed(z):@for(resources(r)|d1(z,r)#EQ#1#AND#ti(z)#NE#0:
   @sum(periods(t):d1(z,r)*X(z,r,t))=tr(z)*or(z))
  );
  @for(periods(t): @for(machines(j):
   @sum(transformed(z):@sum(resources(r):wyk(z,r)*X(z,r,t)))<=1
  ));
  @for(periods(t): @for(resources(r):
   @sum(transformed(z):d(z,r)*X(z,r,t))<=ko(r)
  ));
  @for(technology(z,r,t)|t#GT#1: X(z,r,t-1)-X(z,r,t)<=Y(z,r,t-1));
  @for(transformed(z): @for(resources(r):
   @sum(periods(t):Y(z,r,t))<=1
  ));
  @for(transformed(z): @for(resources(r)|d1(z,r)#EQ#1#AND#ti(z)#NE#0:
   KP(z)=@sum(periods(t):op(t)*Y(z,r,t));
  ));
  @for(transformed(z1): @for(transformed(z2)|do(z1,z2)#EQ#1:
   KP(z2)-or(z2)*ti(z2)>=KP(z1)
  ));
  @for(technology(z,r,t): @bin(X(z,r,t)); @bin(Y(z,r,t)));
  @for(transformed(z): @gin(KP(z)));
 ENDSUBMODEL
 CALC:
  @SET('TERSEO',2);
  @for(periods(t): op(t)=t;
   MAXT=t
  );
  @SOLVE(Constraints, F_objective1);
 ENDCALC
End
```

Algorithm 4: The code in "LINGO" format for the model illustrative after transformation.

(8) $X_{z,r,t} \in \{0,1\}\ \forall z = 1,\ldots,LZ,\ r = 1,\ldots,LR,\ t = 1,\ldots,LT.$

$Y_{z,r,t} \in \{0,1\}\ \forall z = 1,\ldots,LZ,\ r = 1,\ldots,LR,\ t = 1,\ldots,LT.$

$\mathrm{Kp}_z \in C\ \forall z = 1,\ldots,LZ.$

Binary values and integer values.

Description of the Group Predicates

P1: general predicates (universal), independent of the modeled problem (e.g., to create lists based on facts)
P2: predicates that implement the constraints of the problem, objectives, conditions, and so forth, depending on the modeled problem (ID_pro)
P3: predicates that implement different types of questions, depending on the modeled problem (ID_pro)
P4: predicates that implement transformation of the modeled problem, independent of the modeled problem
P5: predicates that implemented the automatic generation of the MILP model.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science, New York, NY, USA, 2006.

[2] Y. Hamadi, E. E. Monfroy, and F. Saubion, *Autonomous Search*, Springer, Heidelberg, Germany, 2011.

[3] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, New York, NY, USA, 1998.

[4] K. Apt and M. Wallace, *Constraint Logic Programming Using Eclipse*, Cambridge University Press, Cambridge, UK, 2006.

[5] M. G. Buscemi and U. Montanari, "A survey of constraint-based programming paradigms," *Computer Science Review*, vol. 2, no. 3, pp. 137–141, 2008.

[6] T. Achterberg, T. Berthold, T. Koch, and K. Wolter, "Constraint integer programming: a new approach to integrate CP and MIP," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, vol. 5015 of *Lecture Notes in Computer Science*, pp. 6–20, Springer, Berlin, Germany, 2008.

[7] A. Bockmayr and T. Kasper, "Branch-and-infer: a framework for combining CP and IP," in *Constraint and Integer Programming*, vol. 27 of *Operations Research/Computer Science Interfaces Series*, pp. 59–87, 2004.

[8] J. N. Hooker, "Logic, optimization, and constraint programming," *INFORMS Journal on Computing*, vol. 14, no. 4, pp. 295–321, 2002.

[9] V. Jain and I. E. Grossmann, "Algorithms for hybrid MILP/CP models for a class of optimization problems," *INFORMS Journal on Computing*, vol. 13, no. 4, pp. 258–276, 2001.

[10] M. Milano and M. Wallace, "Integrating operations research in constraint programming," *Annals of Operations Research*, vol. 175, no. 1, pp. 37–76, 2010.

[11] P. Sitek and J. Wikarek, "A hybrid method for modeling and solving constrained search problems," in *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems (FedCSIS '13)*, pp. 385–392, Kielce, Poland, September 2013.

[12] P. Sitek and J. Wikarek, "A hybrid approach to the optimization of multiechelon systems," *Mathematical Problems in Engineering*, vol. 2015, Article ID 925675, 12 pages, 2015.

[13] P. Sitek, "A hybrid CP/MP approach to supply chain modelling, optimization and analysis," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS '14)*, pp. 1345–1352, Warsaw, Poland, September 2014.

[14] Lindo Systems, *LINDO™ Software for Integer Programming, Linear Programming, Nonlinear Programming, Stochastic Programming, Global Optimization*, 2016, http://www.lindo.com.

[15] SCIP, 2016, http://scip.zib.de.

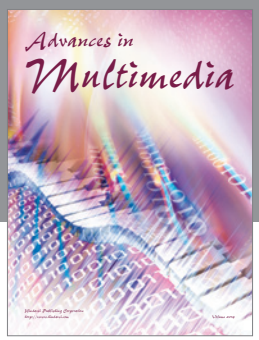
[16] Eclipse, "Eclipse—The Eclipse Foundation open source community website," 2016, http://www.eclipse.org.

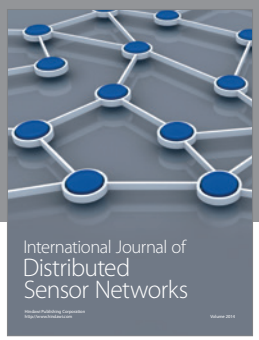
[17] P. Sitek and J. Wikarek, "A novel approach to decision support and optimization of group job handling for multimodal processes in manufacturing and services," in *Proceedings of the 15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM '15)*, pp. 2115–2120, May 2015.

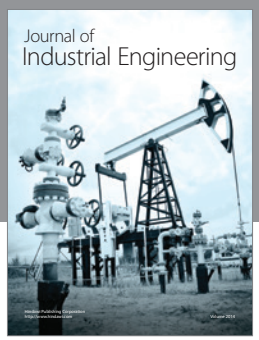
[18] O. Guyon, P. Lemaire, Ă. Pinson, and D. Rivreau, "Solving an integrated job-shop problem with human resource constraints," *Annals of Operations Research*, vol. 213, no. 1, pp. 147–171, 2014.

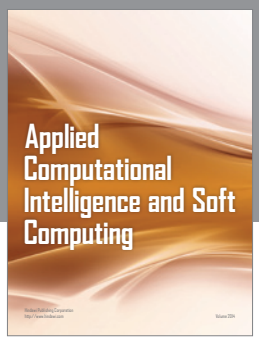
[19] J. Blazewicz, J. K. Lenstra, and A. H. Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983.

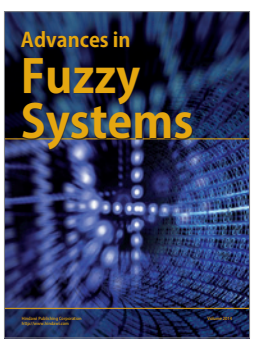
[20] S. Bąk, R. Czarnecki, and S. Deniziak, "Synthesis of real-time cloud applications for Internet of Things," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 23, no. 3, pp. 913–929, 2013.

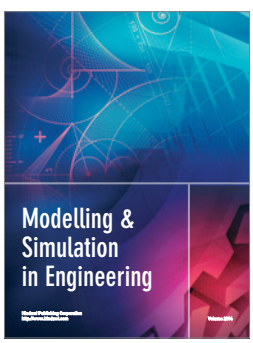


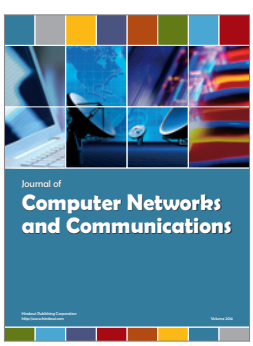












Hindawi
Submit your manuscripts at
http://www.hindawi.com