PAPER *Special Section on Mathematical Systems Science and its Applications*

A Petri Net Approach to Generate Integer Linear Programming Problems

Morikazu NAKAMURA[†a)], **Takeshi TENGAN**[††], ***and*** **Takeo YOSHIDA**[†], ***Members***

SUMMARY This paper proposes a Petri net based mathematical programming approach to combinatorial optimization, in which we generate integer linear programming problems from Petri net models instead of the direct mathematical formulation. We treat two types of combinatorial optimization problems, ordinary problems and time-dependent problems. Firstly, we present autonomous Petri net modeling for ordinary optimization problems, where we obtain fundamental constraints derived from Petri net properties and additional problem-specific ones. Secondly, we propose a colored timed Petri net modeling approach to time-dependent problems, where we generate variables and constraints for time management and for resolving conflicts. Our Petri net approach can drastically reduce the difficulty of the mathematical formulation in a sense that (1) the Petri net modeling does not require deep knowledge of mathematical programming and technique of integer linear model formulations, (2) our automatic formulation allows us to generate large size of integer linear programming problems, and (3) the Petri net modeling approach is flexible for input parameter changes of the original problem.

key words: *mathematical programming, integer linear programming, Petri net, colored timed Petri net, autonomous Petri net*

1. Introduction

Mathematical programming is a potent and useful technique for decision making and problem-solving, which can be applied to a wide variety of real-world problems [1]. Particularly, integer linear programming problems, a class of the mathematical programming formulation, can model many optimization problems [1]–[5]. They can be solved efficiently even for practical size by using optimization tools, so-called solvers, such as CPLEX [6], Gurobi Optimizer [7], and some free software tools.

However, limited users get benefits from the mathematical programming approaches. The main reason should be the difficulty of the mathematical formulation. Users need to formulate their problems as mathematical programming problems, which requires deep knowledge of mathematical programming and enough skills for formulating integer linear programming [1].

Petri nets, a mathematical modeling language, are used to model much variety of concurrent systems such as computer systems, network systems, manufacturing systems, and so on [8]. Petri nets are also a powerful tool for analysis of modeled systems and provide an intuitively understandable graphical presentation of the system's structure and behavior [8], [9]. Once we know some simple rules on Petri nets, we can start system modeling immediately.

This paper presents a methodology to formulate integer linear programming problems based on Petri nets. In our previous letter paper, we reported this approach only for scheduling problems [10]. In this paper, we categorize roughly combinatorial optimization problems into ordinary problems and time-dependent ones, and then propose autonomous Petri net modeling approach to the former type and colored timed Petri net modeling one to the latter type for mathematical formulations. Our proposal can drastically reduce the difficulty of the mathematical formulation of optimization problems.

In [11], the authors proposed a method in which given integer linear programming problems are converted into Petri net models, and then they solve the reachability problem of the converted Petri nets instead of the original ones. It is quite impressive for Petri net research communities, but the direction is opposite from our approach. The objective of our research is to replace direct mathematical formulation by the Petri net modeling. As far as we know, there are no researches for Petri net based methods to generate integer linear programming problems for combinatorial optimization problems.

We also developed a software tool to generate mixed integer linear programming problems from colored timed Petri nets. We employ CPN Tools [12] for Petri net modeling, which can export Petri net models into XML documents. Our methodology is much more useful, compared to the direct formulation with integer linear programming problems.

In this paper, we explain the definitions and basic properties on Petri nets in Sect. 2. We present fundamental constraints derived from basic properties of autonomous Petri nets and show an example of our approach in Sect. 3. In Sect. 4, we propose colored timed Petri net approach for scheduling problems and resource assignments with time constraints. Finally, we conclude with some remarks and future works in Sect. 5.

2. Petri Net Basics

A Petri net is a 4 tuple $PN = (P, T, Pre, Post)$. $P = \{p_1, p_2, ..., p_n\}$ and $T = \{t_1, t_2, ..., t_m\}$ are a set of places and a set of transitions, respectively. A place is denoted by a circle and a transition by a bar. $Pre(p, t)$ and $Post(p, t)$ represent

Manuscript received May 1, 2018.
Manuscript revised September 10, 2018.

[†]The authors are with Computer Science and Intelligent Systems, Faculty of Engineering, University of the Ryukyus, Okinawa-ken, 903-0213 Japan.
[††]The author is with Information Systems, Faculty of International Studies, Meio University, Nago-shi, 905-8585 Japan.
a) E-mail: morikazu@ie.u-ryukyu.ac.jp

DOI: 10.1587/transfun.E102.A.389

the weight on the arc from place p to transition t and from transition t to place p, respectively. A Petri net is called as an *ordinary Petri net* if $Pre(p,t)$ and $Post(p,t)$ are both $\in \{0,1\}, \forall(p,t) \in P \times T$ and as a *generalized* Petri net if $Pre(p,t)$ and $Post(p,t)$ are non-negative integer.

We call p an input place of t when $Pre(p,t) > 0$ and an output place when $Post(p,t) > 0$, respectively. ${}^{\bullet}t$ and $t^{\bullet}$ show the set of the input places and the set of the output places of t, respectively. Similarly, ${}^{\bullet}p$ and $p^{\bullet}$ are the set of input and the set of output transitions of p, respectively.

A marking $M^{tr} = (M(p_1), M(p_2), ..., M(p_n))$ is the vector which represents the number of tokens in each place. Here tr shows the transpose of matrices. A marking M_i shows a token distribution, and it represents a state of the system at a time. Hence, the initial marking M_0 means the initial state of the system.

Transition t is enabled at marking M when $M(p) \geq Pre(p,t), \forall p \in {}^{\bullet}t$ and transition t can be fired when it is enabled. The firing of t removes the same number of tokens as $Pre(p,t)$ from each place $p \in {}^{\bullet}t$, and adds the same number of tokens as $Post(p,t)$ to each place $p \in t^{\bullet}$.

If we regard Pre and $Post$ as a $|P| \times |T|$ matrix, respectively, we can represent the condition for transition t_i to be enabled $x(i)$ times at M as follows:

$$M - Pre \cdot \mathbf{x} \geq \mathbf{0}, \tag{1}$$

where $\mathbf{0}$ is the zero vector of size $|P|$, $\mathbf{x}^{tr} = (x(1), x(2), ..., x(|T|))$ is a vector, called as *firing count vector*, which represents how many times the corresponding transitions are enabled at the current marking M. Moreover, change of a marking can be expressed by the matrix form.

$$M' = M + (Post - Pre) \cdot \mathbf{x} \tag{2}$$

$$M' = M + A \cdot \mathbf{x}, \tag{3}$$

where $A = (Post - Pre)$ is the incident matrix of the Petri net. We call the equation the state equation of the Petri net.

A solution $\mathbf{y}$ of the following equation,

$$\mathbf{y}^{tr} \cdot A = \mathbf{0}, \tag{4}$$

is called as a P-invariant. By multiplying $\mathbf{y}$ to the equation (3),

$$\mathbf{y}^{tr} \cdot M' = \mathbf{y}^{tr} \cdot (M + A \cdot \mathbf{x}) = \mathbf{y}^{tr} \cdot M. \tag{5}$$

This shows that the weighted total sum of marking cannot be changed for the places corresponding to the non-zero elements of $\mathbf{y}$. That is, we can characterize the conservative property based on P-invariants. Similarly, a non-negative solution $\mathbf{x}$ of $A \cdot \mathbf{x} = \mathbf{0}$ is called as a T-invariant.

Colored Petri nets have tokens to which *colors*, showing some attributes, are assigned. Colors of tokens in the input places can be preconditions for firing. On firing, the values of the produced tokens for the output places are calculated based on predefined functions. Therefore, colored Petri nets have mighty modeling power. More details are explained in the literature such as [12].

For quantitative analysis of the dynamical behavior of a system, many researchers introduced *time* to Petri nets [13]–[15]. We can categorize the ways of timing into three types, FD (Firing Duration), HD (Holding Duration), and ED (Enabling Duration) [13]. The FD is to assign time to transitions, where the firing of transition takes time. The HD is referred to as place time Petri nets, where tokens cannot be used for firing for a particular period after located in the place. The last one, the ED, is such that a transition cannot be fired for a given period after enabled. In this paper, we use timed Petri nets with the FD even though we can allow other types in our approach.

Timed Petri nets are a six-tuple $TPN = (P, T, Pre, Post, TS, D)$, where TS is a set of time stamps, usually positive real numbers, and $D : T \rightarrow TS$ is a function to show the firing duration time of transition $t \in T$. A time stamp is also attached to tokens when tokens are generated to record the time. In the timed Petri net, transition t is enabled at time τ when each input place of t has more than or equal to $Pre(p,t)$ tokens and its time stamp is no more than τ. By firing t at time τ, the token distribution should be changed according to the same rule of the Petri net described above except that we attach the time stamp $\tau + D[t]$ to each output token.

Petri nets are *autonomous* if the firing of transitions can be taken place at any timing, that is, in an autonomous manner [9]. Based on this definition, the generalized Petri nets are autonomous. On the contrary, the timed Petri net is a *non-autonomous* Petri net since the system evolution is conditioned by time. Autonomous Petri nets enable us to describe *what happens* only, while timed Petri nets not only *what happens* but also *when it happens*. It is clear that timed Petri nets cannot be converted into autonomous Petri nets.

3. Mathematical Modeling with Autonomous Petri Nets

This section proposes a method to generate integer linear programming problems for ordinary combinatorial optimization problems from autonomous Petri net models.

Firstly, we present fundamental constraints of integer linear programming problems, derived from autonomous Petri net properties. Secondly, as an example, we apply our method to a well-known optimization problem, the traveling salesman problem.

3.1 Fundamental Integer Linear Constraints

We summarize here fundamental integer linear constraints extracted from autonomous Petri nets.

For a given $PN = (P, T, Pre, Post)$ and M_0, we can formulate integer linear constraints.

3.1.1 State Equation and Firing Condition

Let $M_k, k = 1, 2, 3, ..., K$ be markings evolved from M_0 by firing with its firing count vector $\mathbf{x}_k = (x_k(1), x_k(2), ..., x_k(|T|))$,

$$M_{k-1} + (Post - Pre) \cdot \mathbf{x}_k = M_k, k = 1, ..., K \tag{6}$$

$$M_{k-1} - Pre \cdot \mathbf{x}_k \geq 0, k = 1, ..., K \tag{7}$$

$$x_k(i) \in \{0, 1\}, \forall k, i \tag{8}$$

The constraints (6), (7) show the state equation and the firing condition, respectively. We need these constraints when original optimization problems are represented as firing evolutions of Petri nets. K is predetermined by taking the target problems into accounts.

3.1.2 Firing Counts

If we need to restrict such that only one transition can be fired at a time, the following constraint should be required.

$$\sum_{i=1}^{|T|} x_k(i) = 1, \forall k \tag{9}$$

The total firing count for each transition t_i can be limited. If vector $\mathbf{x}'$ is given in which $x'(i)$ shows the desired upper bounds of the firing count for each transition t_i, we can use the following constraint.

$$\sum_{k=1}^{K} x_k(i) \leq x'(i), \forall i \tag{10}$$

3.1.3 Marking Counts

It is not often but sometimes we need to restrict the marking counts by a given vector $\mathbf{m}$.

$$\sum_{k=1}^{K} M_k(i) \leq m(i), \forall i \tag{11}$$

or, in case that the exactly same counts as $\mathbf{m}$ is required,

$$\sum_{k=1}^{K} M_k(i) = m(i), \forall i \tag{12}$$

3.1.4 Boundedness and Safeness

Some essential characteristics of the system behavior are represented by markings. The boundedness constraint requires for place p not to hold tokens more than some specified number U:

$$M_k(p) \leq U, k = 1, ..., K \tag{13}$$

We call a PN with initial marking M_0 to be bounded if the above condition (13) is holding for all the places and all markings from M_0. The bounded Petri nets become a safe Petri net when $U = 1$ for all the places in the condition (13). Note that the boundedness and the safeness described here are behavioral properties but not the structural ones.

3.1.5 Final Marking

The final marking corresponds to the final state of the system. We can restrict the final marking of place p if the desired final state is known. In case that the desired final marking is given as M_f, we need the following constraints:

$$M_K(p) = M_f(p), \forall p \in P \tag{14}$$

Note that for a partial set of places, we can use this constraint.

Moreover, sometimes we want to ensure some places need to include tokens more than or less than a pre-defined value. In this case, we can use '$\geq$' or '$\leq$' in Constraint (14) instead of equality.

3.2 Example

The fundamental integer linear constraints based on properties of autonomous Petri nets are useful for mathematical modeling of ordinary combinatorial optimization problems such as knapsack problems, graph coloring problems, vehicle routing problems, and traveling salesman problems, and so on. Fundamental constraints are used in building blocks for integer linear programming formulation.

As an example, in this section, we show the Petri net based formulation for the traveling salesman problem:

Input: a graph $G = (V, E, d)$ with the set of N cities V, $|V| = N$, the set of directed edges E, where (v_i, v_j) is set if there is a directed path between v_i and v_j, function $d : E \rightarrow \mathcal{R}$ returning the real value distance between the two cities connected by a directed edge, and a city $o \in V$ where the office of the salesman locates.

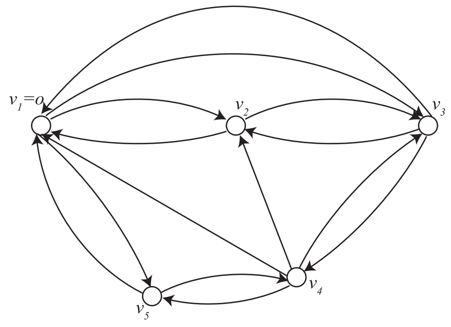
Output: The shortest route such that the salesman starts from o, visits once each city, and returns to o.

In our Petri net modeling, each city v_i corresponds to place P_i, and each directed edge (v_i, v_j) is represented by transition t with two arcs; from P_i to t and from t to P_j. As the distance between v_i and v_j, $d_{i,j} = d(i, j)$, we set the weight $d(t)$ to the corresponding transition t. The initial marking M_0 includes one token in place o and no token in the others.

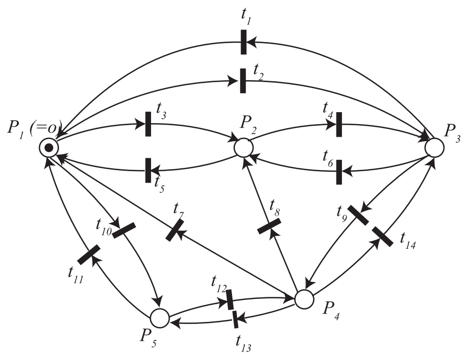
Figure 1 shows an example of five cities problem. For feasible solutions, the Petri net model should evolve from M_0 such that the length of the firing sequence from M_0 is equal to the number of cities plus one, the final marking M_f is the same as M_0, but any other pairs of markings should be different.

Note that only one transition can be fired at any marking M_i since M_0 includes only one token and the number of tokens cannot be changed in any marking thanks to the existence of the P-invariant $\mathbf{y}_I^{tr} = \mathbf{1}^{tr} = (1, 1, ..., 1)$. A weight vector $\mathbf{d}$ for the transitions represents the distance between two cities since the firing of a transition corresponds to the move from the origin city to the destination.

The following formulation can be generated from the fundamental constraints such as the state equation and firing condition, the marking counts, and the final marking constraint. Additionally, we generate the problem-specific objective function.



(a) Graph Instance for Traveling Salesman Problem



(b) Petri Net Model for Traveling Salesman Problem

Fig. 1 Example of traveling salesman problem.

[*TSPIP-PN*]

$$\min \sum_{k=1}^{|P|} \sum_{i=1}^{|T|} x_k(i) \cdot d(t_i) \tag{15}$$

s.t.

$$M_{k-1} + (Post - Pre) \cdot \mathbf{x}_k = M_k, k = 1, \ldots, |P| \tag{16}$$

$$M_{k-1} - Pre \cdot \mathbf{x}_k \geq 0, k = 1, \ldots, |P| \tag{17}$$

$$\sum_{k=1}^{|P|} M_k = \mathbf{1} \tag{18}$$

$$M_0 = M_{|P|} \tag{19}$$

$$M_k(i) \in \{0, 1\}, i = 1, \ldots, |P|, k = 1, \ldots, |P| \tag{20}$$

$$x_k(i) \in \{0, 1\}, i = 1, \ldots, |T|, k = 1, \ldots, |P|, \tag{21}$$

where **1** means ones-vector $(1, 1, \ldots, 1)^{tr}$ of size $|P|$. The constraints (16) and (17) show the state equation constraint and the firing condition constraint. The constraint (18), the marking count constraint, ensures that the salesman visits once each city. The constraint (19), the final marking constraint, confirms the salesman finally returns to o. The objective function (15) denotes the total distance of the obtained route.

The Petri net based formulation for ordinary combinatorial optimization can be performed by building blocks based on fundamental constraints shown in Sect. 3.1.

3.3 Discussion

Integer linear objective functions and constraints can formulate many combinatorial optimization problems. However, users need to know the basic mathematical programming theory even for simple problems. Moreover, it is not straightforward to represent the feasible solution space of the target problem by integer linear objective function and constraints. We sometimes need elaborate techniques for mathematical programming formulation [1].

The following is a well-known integer programming formulation for the traveling salesman problem in the literature [1], where the constraints for avoiding subtours cannot be obtained straightforwardly from the problem definition.

[*TSPIP*]

$$\min \sum_{(i,j)\in E} d_{i,j} \cdot x_{i,j} \tag{22}$$

s.t.

$$\sum_{i,(i,j)\in E} x_{i,j} = 1, \forall j \in V \tag{23}$$

$$\sum_{j,(i,j)\in E} x_{i,j} = 1, \forall i \in V \tag{24}$$

$$\sum_{(i,j)\in E, i\in S, j\in S} x_{i,j} \leq |S| - 1,$$

$$\forall S \subseteq V \setminus \{1\}, |S| \geq 2 \tag{25}$$

$$x_{i,j} \in \{0, 1\}, \forall i, j \tag{26}$$

Binary decision variable $x_{i,j}$ holds 1 if the salesman traverses from city i to j, otherwise 0. The constraints (23) and (24) explain the coming into city j is from one city and the going out from city j is to one city, respectively.

The constraint (25) is for avoiding subtours. Without this constraint, we can have infeasible solutions, that is, the route for the salesman can include disjoint cycles. Note that it is not straightforward to obtain the constraint (25) from the problem definition. We often need such technique for mathematical formulation.

Note also that the number of inequalities for the constraint (25) can be an exponential order. Some researchers reduced the number of constraints for the subtour avoidance successfully [16], [17], but the formulations became more complicated.

The direct formulation of combinatorial optimization requires not only the basic knowledge of the mathematical programming theory but also a technique to represent the feasible solution space by integer linear constraints and the objective function. On the contrary, our Petri net approach for ordinary combinatorial optimization can formulate mathematical programming problems by utilizing fundamental integer linear constraints derived from properties of autonomous Petri nets.

4. Timed Petri Net Modeling for Time-Dependent Optimization Problems

As described in the previous section, autonomous Petri nets are effective to model combinatorial optimization problems for generating integer linear programming problems. However, it is not always useful. For time-dependent optimization problems such as scheduling problems and resource assignments with time constraints, autonomous Petri nets are no longer suitable.

This section considers scheduling problems and resource assignments for flexible manufacturing systems to be modeled with S^4R (Systems of Sequential Systems with Shared Resources) nets. S^4R is a class of Petri nets that is suitable for modeling flexible manufacturing systems [18]–[20]. These problems require to treat *time* to express the duration of tasks. Additionally, to reduce the complexity of modeling, we introduce *colors*. Hence, in this section, we employ colored timed Petri nets for the modeling language for practical time-dependent optimization problems.

The scheduling problems we treat in this paper is defined as follows:

Input: $SP = (TASK, RS, RR, PRE, RT, PT)$, where $TASK$ is the set of tasks, RS the resource set, $RR : TASK \to 2^{RS}$ is the resource requirement of each task, PRE is the precedence relation between tasks, $RT : TASK \to \mathcal{R}$ is the release time of tasks, and $PT : TASK \times RS \to \mathcal{R}$ returns its processing time. These input data satisfy the following conditions.

1. There are multiple sequential systems, and each system includes ordered tasks, that is, the order determines PRE, the precedence relations.
2. Each task can be processed with a single resource type.
3. Multiple resources may be available for each resource type, that is, they have the same functionality but may be different capacities.
4. The processing time for each task with each resource is known in advance, and it is deterministic.

Output: The optimum schedule for processing all tasks, that is, the start time and the end time under the following conditions:

1. No resource can be assigned to more than one task at a time.
2. Resource is always available for processing, that is, *no breakdown*.
3. Operations cannot be interrupted until their completion, that is, *no preemption*.

The scheduling problems with the above conditions can be applied to wide varieties of flexible manufacturing systems [13], [19].

4.1 Modeling with Colored Timed S^4R net

The input data SP specified above are obtained from the domain knowledge of target problems.

In our modeling, we employ a timed Petri net with the FD. That is, each task corresponds to a transition. Each sequential system can be modeled as a state machine of a line structure by inserting a place between adjacent transitions. The places express the state of the process in sequential systems. Secondly, we add a single source p_I and a single sink p_O.

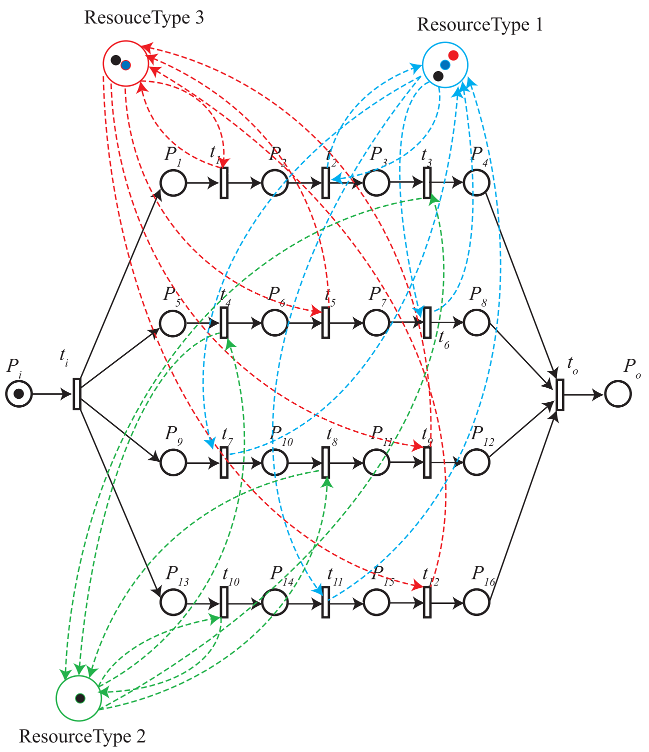


Fig. 2 Colored timed Petri net model for a scheduling problem.

The Petri net model we explained here is called a sound workflow net. The soundness ensures the followings [21]:

1. Only the single source place includes a token at the initial state, and only the single sink place has a token at the final state.
2. All the states reachable from the initial state can lead to the final state.

Figure 2 shows an example of S^4R, where the subnet drawn with black ink represents a sound workflow net.

After modeling all the sequential systems, we add resource places for each resource type, and then connect by an arc from a transition (a task) to the required resource place and vice versa.

In real manufacturing systems, there may be several types of resources. Moreover, even for the same resource type, they can be multiple with different capacities. For representing such characteristics, we introduce colors to the model.

As we described above, there are two types of places in our Petri net models, the places in the workflow subnet and the ones in the resource subnet. Each resource place shown as rp corresponds to a resource type, and it can contain different kinds of tokens, that is, resources with various capacities. For example, suppose we have assembly machines as a resource type. We may be allowed to use one highspeed machine and two middle-speed ones of the assembly machine.

Let RP and WP be the set of resource places and the set of workflow places, respectively. Therefore, the set of places of our Petri net model can be represented by

$$P = RP \cup WP \tag{27}$$

We assign a color to each resource place $rp_i \in RP$ by color function C:

$$C : RP \rightarrow ResourceType \tag{28}$$

$$ResourceType = \{RS_1, RS_1, ...RS_r\} \tag{29}$$

For each place $p \in P$, we locate initial tokens as follows:

$$M_0(p) = \begin{cases} \{UNIT\} & (p = p_I) \\ \emptyset & (p \in WP \setminus \{p_I\}) \\ \{rt_1^p, ..., rt_{n_p}^p\} & (p \in RP) \end{cases} \tag{30}$$

where $UNIT$ shows a token without color, that is, a normal token, and $rt_i^p, i = 1, 2, ..., n_p$ are colored tokens corresponding to resources, and n_p means the number of resources of the resource type $C(p)$.

By referring to the resource requirement of each task, RR, we can connect from each transition $t \in T$ to rp_i, and vice versa. We denote the set of arcs added here by $\hat{Pre}$ and $\hat{Post}$ to differentiate from Pre and $Post$ in the workflow net.

Moreover, let each token have a color representing its capacity to calculate the duration time when the corresponding resource is assigned to a transition:

$$D : T_i \times M_0(rp_i) \rightarrow TS \ \ (\subseteq \mathcal{R}) \tag{31}$$

where T_i is the subset of T such that $\hat{Pre}(rp_i, t) \neq 0, \forall t \in T_i$, $M_0(rp_i)$ is the initial marking of resource place rp_i. The time stamp of all the tokens produced by firing can be calculated by adding this duration time to the start time of the firing. Finally, we obtain a colored timed Petri net for the scheduling problem SP, $CTPN = (P \times RP, T, Pre \cup \hat{Pre}, Post \cup \hat{Post}, TS, D, C)$.

Figure 2 is an example of colored timed S^4R nets, where the subnet drawn with black color shows the workflow net and the subnets colored with blue, green, and red correspond to the resource net. Note that each resource type may have multiple colored tokens, in which different colors in a resource place denote different capacity but the same functionality.

For the scheduling problem, the feasibility of the schedule can be verified with the following conditions [5], [13].

Proposition 1. *A schedule is feasible if and only if the following conditions are satisfied:*

1. *All the precedence relations are satisfied.*
2. *The release time conditions are satisfied.*
3. *There exist no resource conflicts.*

The structure of S^4R provides us the following useful property;

Property 1. *Timed S^4R nets are persistent in a sense that once a transition is enabled, it can be fired sooner or later.*

Proof. S^4R nets may include conflicts for the shared resource usage. When a shared resource is assigned to a task, the task shall return the resource without any consumption so that other tasks can use it. Moreover, the workflow net of S^4R is sound. Therefore, all the transitions can be fired. □

Thanks to Property 1, all we have to do for resolving resource conflicts is to introduce variables and constraints to make the priority between tasks which use the same resource. Moreover, we need to introduce variables to specify which resource to be used for each task since the processing time depends on the capacity of resources. Time-related variables are also necessary, that is, the start time and the end time for each task, where these values cannot contradict the release time, the precedence relations, and the priorities for shared resource usages.

4.2 Generating Mixed Integer Programming Problems

For an obtained colored timed Petri nets $CTPN = (P, T, Pre \cup \hat{Pre}, Post \cup \hat{Post}, TS, D, C)$, release time RT, and M_0, we generate the basic input data for scheduling problem, $SP = (TASK, RS, RR, PRE, RT, PT)$, as follows:

$$TASK = \{1, 2, ..., n\} \leftarrow T = \{t_1, t_2, ..., t_n\} \tag{32}$$

$$RS = \{1, 2, ..., R\}, R = |\cup_{rp \in RP} M_0(rp)| \tag{33}$$

$$RR(j) = \{1, ..., |M_0(rp)|\} \leftarrow \hat{Pre}(t_j, rp) \neq 0, \ \forall t_j \in T \tag{34}$$

$$PRE = \{(i, j) | \text{there exists } p \in P \text{ such that } Pre(p, t_j) \neq 0 \land Post(p, t_i) \neq 0\} \tag{35}$$

$$RT(j) = RT(t_j), \forall j \tag{36}$$

$$PT(j, k) = D(t_j, rt_{r,k}), k \in RR(j), \hat{Pre}(t_j, r) \neq 0, \ \forall t_j \in T \tag{37}$$

4.2.1 Variables

Let us define s_j and e_j to denote the start time and the end time of task $j, \forall j \in TASK$, respectively.

To represent the resource assignment, $\forall j \in TASK, \forall k \in RR(j)$,

$$x_j^k = \begin{cases} 1 & \text{if task } j \text{ is assigned to resource } k \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

To denote the priority of the resource usage among tasks which use the same resource, $\forall (i, j) \in TASK \times TASK$, $RR(i) = RR(j)$,

$$y_{i,j} = \begin{cases} 1 & \text{if tasks } i \text{ and } j \text{ are assigned the same resource and } i \text{ precedes } j \\ 0 & \text{otherwise} \end{cases} \tag{39}$$

4.2.2 Constraints

To enforce that exactly one resource is assigned to each task j, the following constraint is necessary.

$$\sum_{k \in RR(j)} x_j^k = 1, \forall j \in TASK \tag{40}$$

To state the start and the end time, the following constraint should be defined.

$$s_j + \sum_{k \in RR(j)} (PT(j,k) \cdot x_j^k) - e_j = 0, \forall j \in TASK \tag{41}$$

For the precedence relation (j,i), the following constraint is required.

$$s_j + (\sum_{k \in RR(j)} (PT(j,k) \cdot x_j^k)) \leq s_i, \forall (j,i) \in PRE \tag{42}$$

The following constraint ensures that task j starts after task i if $y_{i,j} = 1$.

$$\begin{aligned} & e_i - s_j + U \cdot y_{i,j} \leq U, \\ & \forall i,j \in \{(i,j) | i \neq j, RR(i) = RR(j)\} \end{aligned} \tag{43}$$

where U is a sufficiently large number.

To ensure that the priority variables y are properly set, that is, one of $y_{i,j}$ and $y_{j,i}$ is 1 and the other 0 if tasks i and j are assigned to the same resource, otherwise both of them 0, the following constraints are needed.

$$y_{i,j} + y_{j,i} \leq 1, \forall i,j \in \{(i,j) | i \neq j, RR(i) = RR(j)\} \tag{44}$$

$$\begin{aligned} & x_i^k + x_j^k - y_{i,j} - y_{j,i} \leq 1, \\ & \quad \forall i,j \in \{(i,j) | i \neq j, RR(i) = RR(j)\} \end{aligned} \tag{45}$$

To guarantee that the sequencing variables $y_{i,j}$ and $y_{j,i}$ are zero if tasks i and j are assigned to different resources in the same resource type, the following constraint is necessary.

$$x_i^l + x_j^k + y_{i,j} + y_{j,i} \leq 2, \forall l,k, l \neq k, \forall i,j \tag{46}$$

Finally, the release time constraints can be denoted as follows:

$$s_j \geq RT(j), \forall j \in TASK \tag{47}$$

4.2.3 Objective Function

The objective function will try to minimize the makespan of the schedule even though other objectives are also available.

$$\min \max_{j \in TASK} e_j \tag{48}$$

For formulating as a linear function, we minimize a new variable $emax$ under the following linear constraints:

$$\min emax \tag{49}$$

$$emax \geq e_j, \forall j, (j,i) \notin PRE, \exists i \in TASK \tag{50}$$

4.2.4 Feasibility

The previous subsection explains the variables, the constraints, and the objective function. We can briefly confirm that the mixed integer linear programming problems are valid with respect to **Proposition 1** as follows:

1. All the precedence relations are satisfied with the constraints (41). (42).
2. The release time conditions are satisfied by the constraint (47).
3. There exist no resource conflicts by the constraints (43), (44), (45), and (46).

4.3 Resource Assignments with Time Constraints

Resource assignment problems with time constraints are a scheduling problem in which the objective function is to minimize the total cost to accomplish a given deadline [18].

Input: Addition to the input data for scheduling problems shown above, deadline *Deadline* and a resource cost vector RC, such that $RC(i) \in \mathcal{N}, \forall i \in RS$.

Output: The minimum cost resource assignment so that the makespan of the schedule does not exceed *Deadline*.

For resource assignments with time constraints, we need to change a few points in the mixed integer linear programming problem for the scheduling problem.

Firstly, the deadline constraint can be represented by using variable *emax* in the scheduling problem, constrained by (50).

$$emax \leq Deadline \tag{51}$$

To specify whether or not the schedule plan includes resource i, we introduce an integer vector $\mathbf{z}$, and add the following constraints.

$$z_k \geq x_i^k, \forall i \in TASK, \forall k \in RS \tag{52}$$

Note that this constraints ensure that $z_k \geq 1$ if at least one task is assigned resource k.

Therefore, the following new objective function minimizes the total resource cost:

$$\min RC^{tr} \cdot \mathbf{z} \tag{53}$$

4.4 Software Tool for Generating Integer Linear Programming Problems

We implemented a software tool with Ruby language, where we employ CPN Tools [12] for a Petri net modeling tool. Our Ruby program generates integer linear programming problems from XML documents exported from CPN Tools. It means we can obtain mathematical programming problems

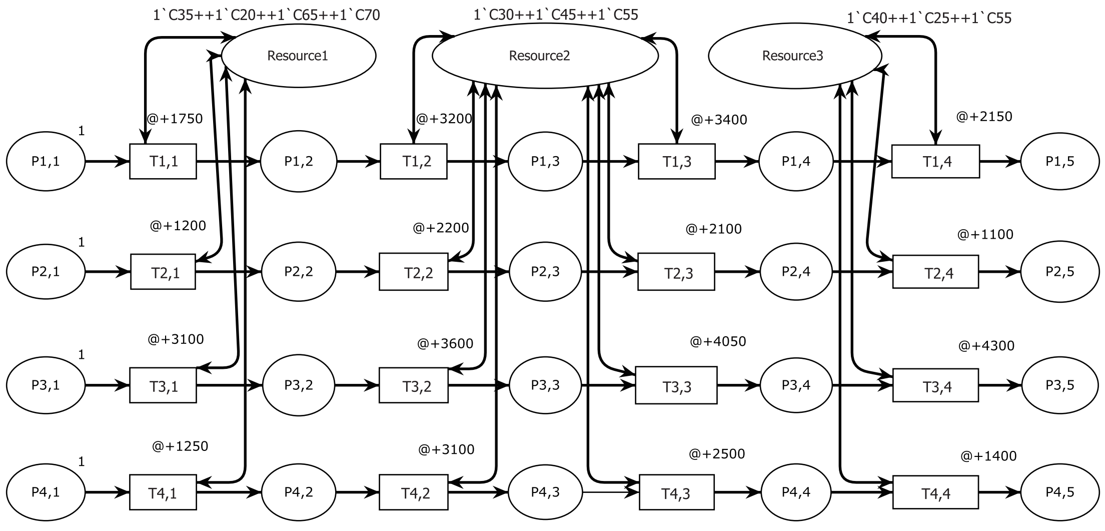

1`C35++1`C20++1`C65++1`C70
1`C30++1`C45++1`C55
1`C40++1`C25++1`C55
Resource1
Resource2
Resource3
@+1750
@+3200
@+3400
@+2150
@+1200
@+2200
@+2100
@+1100
@+3100
@+3600
@+4050
@+4300
@+1250
@+3100
@+2500
@+1400
P1,1 T1,1 P1,2 T1,2 P1,3 T1,3 P1,4 T1,4 P1,5
P2,1 T2,1 P2,2 T2,2 P2,3 T2,3 P2,4 T2,4 P2,5
P3,1 T3,1 P3,2 T3,2 P3,3 T3,3 P3,4 T3,4 P3,5
P4,1 T4,1 P4,2 T4,2 P4,3 T4,3 P4,4 T4,4 P4,5
1

Fig. 3 Colored timed Petri net model for a resource assignment problem using CPN tools 4.0.

Table 1 Petri net model for example.

(a) Place and Initial Marking

Place	*Role*	*Initial Marking*
Resource1	Resource Pool for Task*i*,1	C35, C20, C65, C70
Resource2	Resource Pool for Task*i*,2 &Task*i*,3	C30, C45, C55
Resource2	Resource Pool for Task*i*,4	C40, C25, C55
P*i*,1	Pre condition for Task T*i*,1	*UNIT*
P*i*,2	Post condition for Task T*i*,2	ϕ
P*i*,3	Post condition for Task T*i*,3	ϕ
P*i*,4	Post condition for Task T*i*,4	ϕ
P*i*,5	Post condition for Task T*i*,5	ϕ

C*x* in *Initial Marking* means an attribution of the corresponding token and shows the specification (capacity) of the resource.

(b) Transition (Task Size)

Transition(Task)	*Task Size*[unit]
T*i*,1	1,750, 1,200, 3,100, and 1,250 for $i = 1, 2, 3, 4$, respectively
T*i*,2	3,200, 2,200, 3,600, and 3,100 for $i = 1, 2, 3, 4$, respectively
T*i*,3	3,400, 2,100, 4,050, and 2,500 for $i = 1, 2, 3, 4$, respectively
T*i*,4	2,150, 1,100, 4,300, and 1,400 for $i = 1, 2, 3, 4$, respectively

The average processing time for each task can be determined from the task size and the capacity of the assigned resource, C*x*.

(c) Colored Token (Capacity and Cost)

	C20	C25	C30	C35	C40	C45	C55	C65	C70
Capacity [unit/min.]	20	25	30	35	40	45	55	65	70
Cost	5	10	15	20	30	35	80	200	250

by Petri net modeling without direct mathematical formulation of target problems.

We show an example of a resource assignment problem with time constraints, in which we minimize the total resource cost under the time constraint in task scheduling problems. Figure 3 shows a colored timed Petri net model for manufacturing systems created by CPN Tools. Note that the Petri net model is a S^4R net even though we omitted the source and the sink node in the model. There are four sequential systems and three resource places with initial resource sets (initial marking). Tables 1(a), 1(b), and 1(c) explain the details of the place, the transition sets, and colored tokens, respectively.

We generated by our software tool a mixed integer programming problem from the S^4R net shown in Fig. 3. The generated problem includes 175 variables and 634 constraints. Finally, by using Gurobi optimizer, we solved the problem and obtained the optimum solution. Table 2 shows the results, the total cost and the selected resources, for three cases $Deadline = 900, 600, 300$.

4.5 Discussion

As shown in the example, our method can formulate integer linear programming problems for time-dependent optimization problems such as scheduling and resource assignments by modeling colored timed Petri nets. Our approach and the developed software can be applied to the problems based on

Table 2 Optimum resource assignments (results).

		Resource1				Resource2			Resouce3		
Deadline	*Total Cost*	C35	C20	C65	C70	C30	C45	C55	C40	C25	C55
900	45	✓				✓				✓	
600	80	✓				✓	✓			✓	
300	340			✓		✓	✓	✓			✓

the S^4R class even though we just showed one example due to the space limitation.

The objective of this paper is to propose a Petri net based approach to integer linear programming formulation against the direct mathematical formulation. We summarize here the merits of our approach, compared to the direct formulation, from the viewpoints of *Easiness for Formulation*, *Scalability for Formulation*, and *Flexibility for Parameter Change*:

Easiness for Formulation: Our approach requires only the domain knowledge of the target system and the basic rules of colored timed Petri nets. Once we model the target system with colored timed Petri nets, our tool can generate the corresponding mathematical programming problem. For the direct formulation, we need the knowledge and formulation skills for the mathematical programming in addition to the domain knowledge.

Scalability for Formulation: "Scalability for Formulation" means here how the formulation can be performed when the original problem size becomes large. The number of variables and constraints can often be more than $O(n)$, where n is the target problem size. For example, these numbers for both the resource assignment problems shown in the previous example and the traditional job-shop scheduling problems are proportional to the square of the number of tasks, $O(n^2)$ when we ignore the other input parameters such as the number of resources [22].

The Petri net modeling in our approach is just for the problem domain. Therefore, the modeling effort is only proportional to the problem size, and then our tool can generate all the variables and constraints from the Petri net model. On the other hand, in the direct formulation, we need to generate directly all the variables and the constraints, more than the problem size in many cases. In fact, for the practical size of problems, the number of constraints becomes several hundred, thousands or more. The direct formulation approach is quite difficult to formulate the practical size of integer programming problems correctly.

Flexibility for Parameter Change: Our approach is flexible for changing input parameters since we can generate the new integer linear programming problems just by modifying the Petri net model for the change. For example, when we introduce a new machine to the target production system, all we have to do is just adding one token with the capacity information to the corresponding resource place.

On the contrary, in the direct formulation, we need to add several new variables and many constraints related to the new resource usage. For the case of the resource assignment problem in the previous example or job-shop scheduling problems, the number of new variables and new constraints we need to add is proportional to the number of tasks, $O(n)$.

From these three points of view, compared to the direct mathematical formulation, our Petri net approach can drastically reduce the difficulty of the formulation for time-dependent combinatorial optimization problems.

5. Concluding Remarks

We proposed a Petri net based mathematical programming approach for combinatorial optimization, in which we generate integer linear programming problems from Petri net models instead of the direct mathematical formulation. With autonomous Petri net modeling, we obtain some fundamental constraints for general cases of integer programming problems and show that we can formulate easily ordinary combinatorial optimization problems by our approach. With colored timed Petri net modeling approach, we generate variables and constraints for time management and for serializing tasks for conflict resolving.

The objective of this paper was to propose a new approach to generate integer linear programming problems. Hence, we omitted the evaluation of the computation time because of limited spaces. However, the computation time for optimization depends on solvers and mathematical programming formulations. For future works, we will evaluate the computation time carefully and investigate new efficient constraints.

References

[1] H. Raul Wiliams, Model Building in Mathematical Programming, 5th Edition, Wiley, 2013.

[2] A.B. Keha, K. Khowala, and J.W. Fowler, "Mixed integer programming formulations for single machine scheduling problems," Comput. Ind. Eng., vol.56, no.1, pp.357–367, 2009.

[3] J.C.-H. Pana and J.-S. Chenb, "Mixed binary integer programming formulations for the reentrant job shop scheduling problem," vol.32, no.5, pp.1197–1212, 2005.

[4] D.P. Ronconi and E.G. Birgin, "Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness," Just-in-Time Systems, Springer Optimization and Its Applications, pp.91–105, 2012.

[5] W.-Y. Ku and J.C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," Comput. Oper. Res., vol.73, pp.165–173, 2016.

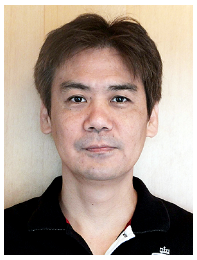
[6] CPLEX Optimizer, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[7] GUROBI Optimizer, http://www.gurobi.com

[8] T. Murata, "Petri nets: Properties, analysis and applications," Proc.

IEEE, vol.77, no.4, pp.541–580, 1989.
[9] R. David and H. Alla, Discrete, Continuous, and Hybrid Petri Nets, Springer, 2005.
[10] A.V. Porco, R. Ushijima, and M. Nakamura, "Automatic generation of mixed integer programming for scheduling problems based on colored timed Petri nets," IEICE Trans. Fundamentals, vol.E101-A, no.2, pp.367–372, Feb. 2018.
[11] A. Kodama and T. Nishi, "Petri net representation and reachability analysis of 0-1 integer linear programming problems," Information Sciences, vol.400-401, pp.157–172, 2017.
[12] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modeling and validation of concurrent systems," Int. J. Softw. Tools Technol. Transfer, vol.9, no.3-4, pp.213–254, 2007.
[13] W.M.P. Van Der Aalst, "Petri net based scheduling," Operations-Research-Spektrum, vol.18, no.4, pp.219–229, 1996.
[14] G. Mušič, "Schedule optimization based on coloured Petri nets and local search," Proc. 7th Vienna International Conference on Mathematical Modeling, Mathematical Modeling, vol.7, Part 1, pp.352–357, 2002.
[15] M.A. Piera and G. Mušič, "Coloured Petri net scheduling models: timed state space exploration shortages," Math. Comput. Simulat., vol.82, no.3, pp.428–441, 2011.
[16] C. Miller, A. Tucker, and R. Zemlin, "Integer programming formulations and traveling salesman problems," J. ACM, no.7, no.4, pp.326–329, 1960.
[17] A.J. Orman and H.P. Williams, "A survey of different integer programming formulations of the traveling salesman problem," Operational Research working papers, LSEOR 04.67. Department of Operational Research, London School of Economics and Political Science, London, UK, 2004.
[18] G. Mejia and C. Montoya, "Applications of resource assignment and scheduling with Petri nets and heuristic search," Ann. Oper. Res., vol.181, no.1, pp.795–812, 2010.
[19] I.B. Abdallah, H.A. Elmaraghy, and T. Elmekkawy, "Deadlock-free scheduling in flexible manufacturing systems using Petri nets," Int. J. Prod. Res., vol.40, no.12, pp.2733–2756, 2002.
[20] H. Lei, K. Xing, L. Han, F. Xiong, and Z. Ge, "Deadlock-free scheduling for flexible manufacturing systems using Petri nets and heuristic search," Comput. Ind. Eng., vol.72, pp.297–305, 2014.
[21] W.M.P. van der Aalst, "The application of Petri-nets to workflow management," J. Circuit. Syst. Comp., vol.8, no.1, pp.21–66, 1998.
[22] E. Driss, R. Mallouli, and W. Hachicha, "Mixed integer programming for job shop scheduling problem with separable sequence-dependent setup times," American Journal of Mathematical and Computational Sciences, vol.3, no.1, pp.31–36, 2018.

Morikazu Nakamura received the B.E. and M.E. degrees from University of the Ryukyus in 1989 and 1991, respectively, and D.E. degree from Osaka University in 1996. He is currently a professor in Area of Computer Science and Intelligent Systems, Faculty of Engineering, University of the Ryukyus. His research interests include theory and applications on mathematical systems. He is a member of IEEE.



Takeshi Tengan received the B.E. and M.E. degrees in Electronics and Information Engineering from University of the Ryukyus in 1994 and 1996, respectively. He is currently a senior associate professor in Information Systems Major, Faculty of International Studies, Meio University. His research interests include optimization and soft computing. He is a member of the Japanese Society of Evolutionary Computation.

Takeo Yoshida received the B.E. and M.E. degrees in electrical engineering from Nagaoka University of Technology and the D.E. degree in electrical engineering from Tokyo Metropolitan University in 1991, 1993 and 1997, respectively. He is currently an assistant professor in the Department of Engineering, University of the Ryukyus. His research interests include dependable computing, VLSI design, and graph theory. He is a member of IEEE and IPSJ.