

Dominance rules in combinatorial optimization problems

Antoine Jouglet & Jacques Carlier

UMR CNRS 6599 Heudiasyc, Université de Technologie de Compiègne
Centre de Recherches de Royallieu, 60200 Compiègne, France
antoine.jouglet@hds.utc.fr, jacques.carlier@hds.utc.fr

Abstract

The aim of this paper is to study the concept of a “dominance rule” in the context of combinatorial optimization. A dominance rule is established in order to reduce the solution space of a problem by adding new constraints to it, either in a procedure that aims to reduce the domains of variables, or directly in building interesting solutions. Dominance rules have been extensively used over the last fifty years. Surprisingly, to our knowledge, no detailed description of them can be found in the literature other than a few short formal descriptions in the context of enumerative methods. We are therefore proposing an investigation into what dominance rules are. We first provide a definition of a dominance rule with its different nuances. Next, we analyze how dominance rules are generally formulated and what are the consequences of such formulations. Finally, we enumerate the common characteristics of dominance rules encountered in the literature and in the usual process of solving combinatorial optimization problems.

Keywords: combinatorial optimization, dominance rules, constraints, modeling

1. Introduction

In combinatorial optimization, the term “dominance rule” seems to have been introduced in the late fifties. We can find it in early papers by Manne [1], Levitan [2], Ignall and Schrage [3], Fox [4], Proshan and Bray [5], Weingartner and Ness [6] and Elmagharaby [7], although the term seems have taken several years to be established. Later, it was intensively used by Baker [8], Kohler and Steiglitz [9] and Ibaraki [10]. Dominance rules had, however, been used much earlier, but under different names. Johnson [11] talked about a “decision rule” for the two-machine flowshop scheduling problem, since such a rule allowed him to construct an optimal solution directly. In the same way, Smith [12] spoke of a “sufficient condition of optimality” for the one-machine total weighted completion time scheduling problem. In the context of integer linear programming, Gomory [13] used the term “additional constraints” that are needed to be satisfied by integer solutions. Swift [14] talked about a “rejection” rule for eliminating solutions isomorphic to other solutions. Giffler and Thompson [15] spoke of a characterized subset of solutions containing a subset of optimal solutions.

These different terms are often contextual but express the same general idea: eliminating uninteresting solutions, or selecting interesting solutions. The common theme is that one subset of solutions is rejected while the complementary subset is retained.



The interest of dominance rules is obvious, since their *raison d'être* is to reduce, either statically or dynamically, the search space of combinatorial problems which are in the process of being solved. Among these, scheduling problems have been the subject of much investigation because of their applications in industry. "Scheduling is the allocation of resources over time to perform a collection of tasks" [8]. Since the early works of Johnson [11] and Smith [12] in the mid fifties, scheduling has remained at the interface between theory and practice [16]. It has given rise to a number of techniques for problem solving including dominance rules which have played an important role. Therefore, throughout this paper, we shall be looking at a single-machine scheduling problem which is sufficiently general to illustrate all the concepts.

In this paper we try to provide a kind of bird's eye view of dominance rules in combinatorial optimization problems. In Section 2 we define combinatorial optimization problems and provide definitions which will be useful later on. In Section 3 we define the concept of a dominance rule. In Section 4 we analyze how dominance rules are generally formulated and what the consequences of such formulations are. Next, in Section 5 we provide a list of the main characteristics of dominance rules encountered in the literature. In Section 6 we show how the concept of dominance is extended to dominance rules themselves, or to problems or instances of problems. Finally, the main uses of dominance rules for solving combinatorial optimization problems are discussed in Section 7.

2. Combinatorial optimization problems

In this section, we begin by defining what we mean by combinatorial optimization problems and we provide definitions which will be useful later on. In particular, the notions of a problem, of an instance and a solution of a problem, and of the procedures of modeling and solving a problem are discussed.

2.1. Notion of problem: formalizing a question to solve

A problem can be defined as "**a question to solve**". A large number of ("real" or "invented") problems are from the domain of the computation and can be handled by using computers. For Garey and Johnson [17], a **problem** is a general question to be answered. It usually possesses several **parameters**. A problem is described by giving: (1) a general description of all its parameters, and (2) a statement of what properties a **solution** of the problem is required to satisfy. A problem can be viewed simply as a "**basic model**", *i.e.* a simple formalization, the mere comprehension of which suggests a set of solutions.

For example, scheduling problems are described by giving the characteristics of the available resources and of the set of tasks. In particular, a single-machine problem is a problem in which only one machine which can process only one task at a time (the machine is said to be disjunctive and the tasks to be subject to disjunctive constraints) is available from time 0. Each task i has a processing time p_i and a release date r_i before which the job cannot be processed by the machine. The preemption of tasks is not allowed, meaning that a task i has to be processed by the machine without interruption over p_i units of time from its starting time to its completion time. A question to be answered is, for example, whether a starting time can be assigned to each task such that the sum of the task completion times $\sum_i C_i$ is minimal. Following the 3-field notation widely used in the literature (see for example [18]), this problem will be denoted as $1|r_i|\sum C_i$ throughout this paper. Moreover, the special case of this problem

in which all release dates are equal to 0 is denoted as $1||\sum C_i$. Suppose we associate a due date d_i with each task i , *i.e.* a date before which we would like i to be completed. Suppose also that if a task i is completed after d_i it is considered to be late and that its tardiness T_i is defined as being equal to the difference between its completion time C_i and its due date d_i , *i.e.* $T_i = \max(0, C_i - d_i)$. Another question is whether a starting time can be assigned to each task such that the sum of the task tardiness is minimal. This problem is denoted as $1|r_i|\sum T_i$, or $1||\sum T_i$ when all release dates are equal to 0. Finally, suppose that we associate a deadline \bar{d}_i with each task i , *i.e.* a date before which task i has to be strictly completed. Another question might then be whether a starting time can be assigned to each task such that each task starts after its release date and is completed before its deadline. This problem is denoted as $1|r_i, \bar{d}_i|_-$.

2.2. Instances of a problem

An **instance** of a problem is obtained by specifying particular values for all the problem parameters [17]. This description of a problem instance can be viewed as a finite string of symbols chosen from a finite input alphabet according to one particular fixed **encoding scheme**. The **input length** for an instance of the problem is then defined as the number of symbols in the description of the instance and it is used as the formal measure of instance size. In the case of a single-machine problem, an instance is specified by its number of tasks and by the values given to the processing times, the release dates, the due dates and the deadlines of the tasks if they are required by the studied version of the problem. The instance size is generally expressed as a multiple of the number of tasks, since the maximum number of symbols used for each value is generally considered to be fixed.

Note that since an instance has to be decoded, we can complete Garey and Johnson's [17] definition of a problem by adding a polynomial time **decoding program** of fixed size which is used to read an instance (a string of symbols) of variable size and to store its parameters in the memory of the computer.

2.3. Modeling a problem

As we have said above, once a problem has been put into words, it can be viewed as a basic model formalizing the question to solve. Now it has to be modeled. **Modeling** means elaborating a more sophisticated model than the basic model. This new model will allow us to solve the problem. Modeling consists of identifying the objective function, the variables, and the constraints for the given problem. This step is not neutral as regards to the eventual solution. Formulating a "good" model is of crucial importance in solving the problem" [19]. Indeed, it is at this stage that the **decision variables** of the problem are defined. Most of the time several possibilities occur and the variables are often chosen simply from the definition of a solution. For example, the variables of the single-machine scheduling problem described above could be the completion times of the tasks. However, an obvious choice may not be a good one when it comes to solving the problem. Thus, the choice between the different ways of modeling has a dramatic impact on the effectiveness of its solution. In particular, the **structural properties** of the solutions to be obtained have to be expressed via the model. For example, another model of the single-machine scheduling problem could be obtained by considering only one variable corresponding to the sequence in which tasks have to be processed. While this model necessitates a procedure to compute the starting times of the tasks associated with a sequence, the disjunctive constraints of the machine are easier to express in this model and lead to more effective solving methods in a lot of cases.

Once modeled, a **combinatorial optimization problem** can then be formally defined by:

- a n -tuple of **variables** (x_1, x_2, \dots, x_n) , the value of each variable x_i belonging to a set $D(x_i)$ that we call the **domain** of x_i ;
- a set C of **constraints** on variables;
- an **objective function** $\Phi : D(x_1) \times D(x_2) \times \dots \times D(x_n) \rightarrow \mathbb{R}$ which associates a value with each assignment.

Hence, a combinatorial optimization problem can be viewed as a second, more sophisticated model. In contrast to the “basic model”, which only formalized the question to solve, the aim of this new model is to solve the problem. Given a set of parameters of the basic model obtained by decoding an instance of the problem, a second polynomial program (in addition to the first program that was used to read and decode an instance), must exist to obtain a new set of parameters for the combinatorial optimization problem. Note that a large range of flexibility is permitted in both the constraint set and the objective function. Note also that while several problems are **multi-criteria**, this paper only considers problems with one objective function at a time.

The constraints of a combinatorial problem express which combinations of variable values are allowed. A constraint can be formally defined as follows:

Definition 1. A **constraint** c belonging to C is a function $c : \Omega = D(x_1) \times D(x_2) \times \dots \times D(x_n) \rightarrow \mathcal{P}(\Omega)$, which associates with Ω a subset $c(\Omega) \subseteq \Omega$ containing the assignments satisfying the constraint. For the constraint c , an element belonging to subset $c(\Omega)$ is said to be *feasible*, and an element belonging to its complementary with respect to Ω , i.e. $\Omega - c(\Omega)$, is said to be *unfeasible*.

The conjunction of all subsets defined by the constraints belonging to C defines the set of feasible solutions of the problem.

From now on, the modeling of the “real” problem to be solved is assumed to be fully specified, and the term “problem” refers to a combinatorial search problem as defined above. Thus, the single-machine problem with n tasks can be formally defined by the n -tuple (C_1, \dots, C_n) of completion times with $D(C_i) = \{r_i + p_i, r_i + p_i + 1, \dots, \bar{d}_i\}$. Note that in a problem with no deadline, the \bar{d}_i values can be assumed to be a large number. A disjunctive constraint between two tasks i and j , $i \neq j$, is expressed as follows: $(C_i \geq C_j + p_i) \vee (C_j \geq C_i + p_j)$. The entire set of disjunctive constraints between the different pairs of tasks has to be satisfied. Note that constraints such as $C_i \geq r_i + p_i$ and $C_i \leq \bar{d}_i$ are also taken into account in the model through the domain variables. The objective function to consider in the example is the total completion time $\sum_i C_i$ or the total tardiness $\sum_i T_i$.

2.4. Solving a problem

Once modeled, solving a **combinatorial optimization problem** consists of assigning a **value** to each variable. An **assignment** has to be chosen from the Cartesian product $\Omega = D(x_1) \times D(x_2) \times \dots \times D(x_n)$ in such a way that:

1. the set C of **constraints** is satisfied;
2. the **objective function** Φ is minimized.

Note that maximizing an objective function is equivalent to minimizing the negative of the same function. Thus, the above formulation also covers maximization problems.

In this paper we consider that a **combinatorial optimization problem** is any problem as defined above which is decidable, *i.e.* for which there exists an algorithm solving the problem in a finite number of steps. Of course, if Ω is finite, the problem is decidable.

2.5. Optimization problems and search problems

We wish to obtain an assignment satisfying all the constraints and which minimizes the objective function Φ . Thus, an objective function is required to associate with each assignment a value from among a given set of values referred as $D(\Phi)$. We extend the objective function to subsets of solutions: for any subset S of solutions, we define $\Phi(S) = \min_{z \in S} \Phi(z)$. The existence of an assignment satisfying the constraints of the problem is usually not part of the question. Often, the number of such possible assignments is irrelevant and all that matters is to find an optimal assignment. That is why we generally speak of “**optimization problems**”.

The objective function can be bi-valued by 0 and ∞ , corresponding to feasible (the constraints are eventually satisfied) and unfeasible (at least one constraint is not satisfied), and it may not be known in advance whether or not feasible solutions exist. Therefore, the problem consists in finding a feasible solution or concluding that no such solution exists. Here the problem is referred as a “**search problem**” rather than an “optimization problem”. For example, searching for a feasible solution to a single-machine scheduling problem when deadlines are considered (*i.e.* $1|r_i, \bar{d}_i|_$) is such a problem. Nevertheless, the formulation of a combinatorial optimization problem, as described above, is often used to cover this case as well as the multi-valued or continuous valued objective function [20, 21]. For the sake of simplicity we will adopt this convenient generalization in this paper, while noting that optimization problems are also often solved using successive solutions of search problems.

Nevertheless, in the following pages we shall sometimes need to distinguish between what derives from “feasibility” considerations (in relation to a set of constraints C), and what derives from “optimality” considerations (in relation to the optimization of an objective function Φ).

2.6. Constraints

The constraints of a combinatorial search problem can be implicit or explicit (see for example [21]). **Implicit constraints** are those which are implicitly satisfied by the way the problem has been modeled. Thus they do not require an explicit procedure to ensure that they are satisfied, because they are satisfied according to the variables and the domain of the variables which have been chosen to model the problem. **Explicit constraints**, on the other hand, will require procedures for recognition as part of the method employed to solve the problem. For example, in the model described above for the single-machine scheduling problem, constraints $C_i \geq r_i + p_i$ and $C_i \leq \bar{d}_i$ are implicit because they are always satisfied through the domain $D(C_i) = \{r_i + p_i, r_i + p_i + 1, \dots, \bar{d}_i\}$ of variable C_i . However, disjunctive constraints are explicit and require a procedure to determine whether they are satisfied or not.

A constraint can be **intentional**, *i.e.* expressed using a relation between variables which has to be satisfied, *e.g.* an arithmetic formula. It can also be **extensional**, *i.e.* expressed as a set of tuples of values that satisfy the constraint.

In constraint programming the **global constraints** are usually identified (see for example [22]). A global constraint corresponds to a conjunction of constraints which it is more convenient to consider together, since this leads to greater clarity, in addition to improving the problem solving process. For example, in the single-machine scheduling problem, it is common to consider the set of disjunctive constraints as only one global constraint. Indeed, some efficient techniques known as Edge-Finding (see for example [23]) allow more deductions to be obtained when considering disjunctive constraints all together.

In light of our remarks made in the previous section, we might also wish to identify the set of “**feasibility constraints**” of the problem, which correspond to the constraints cited above, as well as the “**optimality constraint**”, which stipulates the requirement for solutions having the best value with respect to the objective function.

2.7. The notion of a solution

We wish to determine a vector from the Cartesian product space $\Omega = D(x_1) \times D(x_2) \times \dots \times D(x_n)$ which minimizes the objective function Φ and satisfies all the constraints.

A partial assignment where a value is specified for some variables of the problem while the values of others remain unknown is called a **partial solution**.

Every complete assignment belonging to Ω is called a **solution**, since it satisfies the implicit constraints [21]. It does not matter whether or not the assignment is feasible with respect to the explicit constraints. It is usual to designate Ω as being the **solution space**. In the following we shall refer to the power set of Ω , *i.e.* the set of all subsets of Ω , as $\mathcal{P}(\Omega)$.

For a lot of problems we may consider that each selection space $D(x_i)$ contains only a finite number of distinguishable (numeric or symbolic) values. In this case we refer to the cardinal of domain $D(x_i)$ as $|D(x_i)|$. The solution space is then also finite and its cardinal is equal to $|\Omega| = \prod_{i=1}^n |D(x_i)|$.

A solution respecting all the constraints belonging to C is called a **feasible solution**. From now on we shall refer to the subset of feasible solutions belonging to Ω as $\mathcal{F}(\Omega)$. Any solution which does not belong to $\mathcal{F}(\Omega)$ is said to be **unfeasible** or **unacceptable**. The subset of unfeasible solutions is referred as $\mathcal{U}(\Omega)$. Note that $\mathcal{F}(\Omega)$ and $\mathcal{U}(\Omega)$ are disjoint and complementary according to Ω , *i.e.* $\mathcal{F}(\Omega) \cap \mathcal{U}(\Omega) = \emptyset$ and $\mathcal{F}(\Omega) \cup \mathcal{U}(\Omega) = \Omega$. We also define $\mathcal{F}(S)$ as the subset of feasible solutions of any subset S of Ω , *i.e.* if $S \subseteq \Omega$ then $\mathcal{F}(S) = \mathcal{F}(\Omega) \cap S$. $\mathcal{U}(S)$ is defined as the set of unfeasible solutions of the set S .

Any feasible solution $z^* \in \mathcal{F}(\Omega)$ such that $\forall z \in \mathcal{F}(\Omega), \Phi(z^*) \leq \Phi(z)$ is called an **optimal solution**. The value $\Phi(z^*) = \Phi(\mathcal{F}(\Omega))$ is called the **optimal value** of the problem. In the following, the subset of optimal solutions of Ω is referred to as $\mathcal{O}(\Omega)$. We have $\mathcal{O}(\Omega) \subseteq \mathcal{F}(\Omega)$. In the case of a “decision” problem, subsets $\mathcal{O}(\Omega)$ and $\mathcal{F}(\Omega)$ coincide, *i.e.* $\mathcal{O}(\Omega) = \mathcal{F}(\Omega)$. Any feasible solution which is not in $\mathcal{O}(\Omega)$, *i.e.* any solution belonging to $\mathcal{F}(\Omega) - \mathcal{O}(\Omega)$ is said to be **suboptimal**. We also define $\mathcal{O}(S)$ as the subset of the optimal solutions of any subset S of Ω . Thus, $z \in \mathcal{O}(S)$ if and only if $\Phi(z) = \Phi(S)$. Note that $\Phi(S)$ can be greater than $\Phi(\Omega)$, *i.e.* none of the optimal solutions of Ω belongs to S .

2.8. Solving combinatorial search problems

For Garey and Johnson [17], algorithms are general step-by-step procedures for solving problems. An algorithm is said to **solve** a problem Π if that algorithm can be applied to any instance I of Π and is always guaranteed to produce an optimal solution for this instance I .

There are many algorithms available for solving problems. While some algorithms are dedicated to specific problems, others are more generic methods suitable for solving more

than one problem. Many methods are appropriate only for certain types of problems. That is why we have to recognize the characteristics of a problem in order to identify an appropriate technique for solving it. Optimization problems are then classified according to their characteristics (the nature of the objective function, variables and constraints). Thus, algorithms are often classified according to the type of problems they are designed to solve, or according to whether they are designed to produce optimal solutions or only approximate (but possibly optimal) solutions. Among these general techniques, Linear Programming, Integer Linear Programming, Dynamic Programming, Backtrack Programming and Constraint Programming are the most frequently used and the most effective techniques for dealing with search problems. Where there is no possibility (owing to the size or the intractability of the problem) of obtaining an optimal solution, or no requirement for the solution to be optimal, then suboptimal solutions only may be sought, using approximate algorithms called heuristics.

It is interesting to note that the effectiveness of the solving methods (other than in a few specific cases of approximate methods) is highly dependent on the possibility of finding structural properties of feasible and optimal solutions which can then be used to identify the solutions of interest. Most of these properties can be stated as "dominance rules" whose *raison d'être* is discerning interesting subsets of solutions in which it is sufficient to search for optimal solutions.

3. Dominance rules

In this section, we provide a formal definition of the dominance rule concept with its different nuances.

Few definitions of dominance rules are to be found in the literature. Baker and Trietsch [16] define them in the following way: "*Dominance properties provide conditions under which certain potential solutions can be ignored*".

One formal definition that we encounter is by Carlier and Chrétienne [24]. We also find formal definitions by Kohler and Steiglitz [9, 25], and also by Ibaraki [10], whose definitions pertain to dominance relations (see Section 5.1). All these definitions are provided in the specific context of enumerative methods. We now put forward the following definitions in the general context:

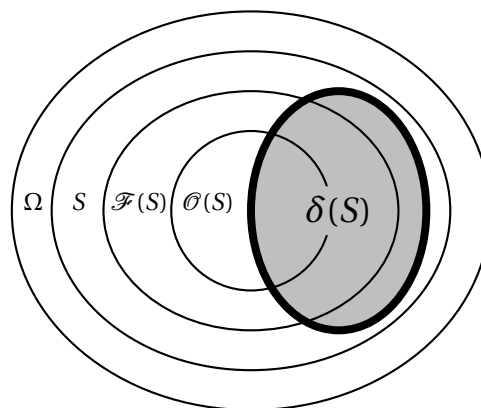


Figure 1: $\delta(S)$ is dominant and $S - \delta(S)$ is dominated.

Definition 2. A **dominance rule** $\delta : \mathcal{P}(\Omega) \rightarrow \mathcal{P}(\Omega)$ associates with a given subset $S \subseteq \Omega$ a subset $\delta(S) \subseteq S$ such that $\mathcal{F}(S) \neq \emptyset \Rightarrow \mathcal{O}(S) \cap \delta(S) \neq \emptyset$. The subset $\delta(S)$ is said to be a **dominant** subset of S and its complementary with respect to S , i.e. $S - \delta(S)$, is said to be **dominated**.

Informally, a dominance rule identifies a subset of S containing at least one optimal solution of S (see Figure 1). However, when the dominant part includes all optimal solutions, we can refer to a strict dominance rule, since all solutions belonging to $\mathcal{O}(S) \cap \delta(S)$ are then strictly better than solutions belonging to $S - \delta(S)$ (see Figure 2):

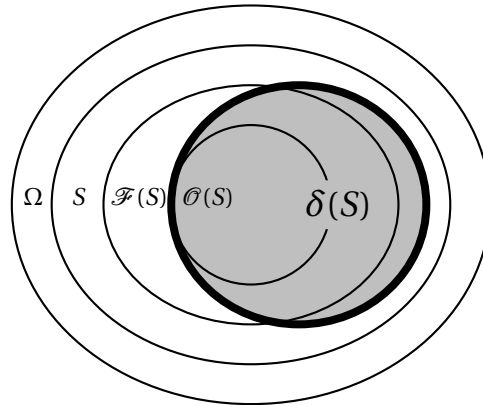


Figure 2: $\delta(S)$ is strictly dominant and $S - \delta(S)$ is strictly dominated.

Definition 3. A **strict dominance rule** $\delta : \mathcal{P}(\Omega) \rightarrow \mathcal{P}(\Omega)$ associates with a given subset $S \subseteq \Omega$ a subset $\delta(S) \subseteq S$ such that $\mathcal{F}(S) \neq \emptyset \Rightarrow \mathcal{O}(S) \subseteq \delta(S)$. The subset $\delta(S)$ is said to be a **strict dominant** subset of S and its complementary with respect to S , i.e. $S - \delta(S)$, is said to be **strictly dominated**.

Moreover, when the dominant part includes all feasible solutions, we can refer to a redundant dominance rule since it does not eliminate feasible solutions (see Figure 3):

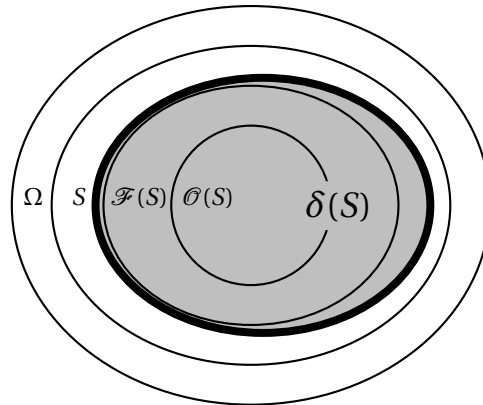


Figure 3: $\delta(S)$ is redundantly dominant and $S - \delta(S)$ is redundantly dominated.

Definition 4. A **redundant dominance rule** $\delta : \mathcal{P}(\Omega) \rightarrow \mathcal{P}(\Omega)$ associates with a given subset $S \subseteq \Omega$ a subset $\delta(S) \subseteq S$ such that $\mathcal{F}(S) \subseteq \delta(S)$. The subset $\delta(S)$ is said to be a **redundant dominant** subset of S and its complementary with respect to S , i.e. $S - \delta(S)$, is said to be **redundantly dominated**.

Most of the time δ is expressed using a property A which is satisfied by optimal or feasible solutions (see Section 4). Moreover, if $\mathcal{O}(\Omega) \cap S \neq \emptyset$ then at least one optimal solution of the problem belongs to $\delta(S)$. In this case $\delta(S)$ is also a dominant subset of Ω .

In scheduling, a semi-active schedule is defined as a schedule in which no job can be scheduled earlier without changing the sequence of execution of jobs on the machine or violating the constraints [18]. In other words, a semi-active schedule is a schedule in which no local left-shift is possible (i.e. a schedule containing no superfluous idle time). The application which associates the set of semi-active schedules with the set of all feasible schedules is a dominance rule, since there is at least one optimal schedule which is semi-active [8]. Note that this dominance rule is strict when considering the total completion time criterion ($|r_i| \sum C_i$) since all optimal schedules are semi-active (a schedule with a superfluous idle time cannot be optimal because a simple local left shift of the task scheduled just after the idle time allows the value of the objective function to decrease strictly). However, it is not the case when considering the total tardiness criterion. For example, consider an optimal schedule of an instance of $|r_i| \sum T_i$ in which the last task of the schedule is strictly completed before its due date. This task can then be delayed until its due date without changing the optimality of the schedule while it is no longer semi-active. Note also that dominance rules on which Edge-Finding techniques are based are redundant because they filter the set of solutions by eliminating only solutions which are not feasible according to the disjunctive constraints.

In the above definitions, it is not required that the dominance rules provide an operational method for building such a subset $\delta(S)$. A dominance rule is established in order to reduce the solution space either by adding new constraints to the problem, or by writing a procedure that attempts to reduce the domains of variables, or by building interesting solutions directly. Note that algorithms are not themselves dominance rules: they simply make use of them. The process of filtering the domain of a variable using a dominance rule is called an **adjustment**. In constraint programming it is also called **domain reduction**. An algorithm performing one or several adjustments is called a **filtering algorithm**.

In fact, most of the time, procedures or added constraints using dominance rules to reduce the search space are not able to eliminate the dominated part totally. The result obtained is therefore a subset of S containing $\delta(S)$. Nevertheless, note that the following proposition obviously holds:

Proposition 1. Let S be a subset of Ω such that $\mathcal{F}(S) \neq \emptyset$. Given $\delta(S)$, a dominant subset of S , any subset S' of S such that $\delta(S) \subseteq S'$ is a dominant subset of S .

Proof. Since, by definition, $\delta(S) \cap \mathcal{O}(S) \neq \emptyset$, and $\delta(S) \subseteq S'$, then $\delta(S) \cap \mathcal{O}(S) \subseteq S' \cap \mathcal{O}(S) \neq \emptyset$. \square

The choice of the dominance rule δ usually involves a compromise between a powerful expression and an easy identification of solutions belonging to $\delta(S)$.

When using a dominance rule, the considered subset of solutions S depends on the algorithm in which the dominance rule is used. However, most of the time such a subset is represented via a partial solution: the subset S contains all solutions which can be derived from this partial solution according to the domains of the unfixed variables.

If we carry the above definition to its limits we can think of dominance rules as constraints of the problem (feasibility constraints as well as optimality constraints). Indeed, constraints intentionally define a subset of solutions containing optimal solutions. Of course, by definition $\mathcal{O}(\Omega)$ is a dominant set of Ω . If $\mathcal{O}(\Omega)$ can be built directly, the problem is solved. In fact, it is more interesting to think of constraints as dominance rules only if they can be used in an operational way, *i.e.* if they lead to procedures or new constraints that can reduce the search space as the computation progresses, and not only to test the validity of a given solution (which may or may not be a partial solution). Note that dominance rules can often be put into practice by the addition of new constraints to the initial set of constraints.

4. Formulations of dominance rules

In this section, we analyze how dominance rules are generally formulated and what the consequences entailed by such formulations are. Most of the time a dominance rule is expressed using a property A which allows a dominant subset and a dominated subset of S to be identified. Formulation can be done in different ways. The possible adjustments which can be performed according to the dominance rule being considered are highly dependent on the characteristics of property A and on the formulation which uses this property.

Note that a large range of flexibility is permitted in the expression of A . Thus, using a dominance rule involves placing oneself at the mercy of the property A , since identifying solutions satisfying A may or may not be straightforward.

We now analyze different formulations of a dominance rule according to a given property A and we show what kind of adjustments can be performed according to a formulation. Consider a subset S of Ω . When we refer below to "an optimal solution of S with respect to S " we mean a solution z such that $z \in \mathcal{O}(S)$, *i.e.* such that $\Phi(z) = \Phi(S)$.

4.1. Universal property-based dominance rule

This dominance rule is expressed as "**all optimal solutions of S with respect to S have the property A** ". Therefore A is a **necessary condition of optimality with respect to S** . The subset of solutions belonging to S and satisfying A is dominant. Note that some solutions satisfying A may not be optimal. From a practical point of view:

- Any solution which does not satisfy A can be removed from S since it is not optimal.
- We can retain any identified subset of S containing the whole subset of solutions satisfying A , since this superset of A is dominant (see Proposition 1).

We refer to such a formulation as a **universal property-based dominance rule**. For example, "given an instance of $1|r_i|\sum C_i$, all optimal schedules are semi-active" is such a rule [8]. Thus, only semi-active schedules need to be considered. This dominance rule allows the use of a model that considers sequences instead of completion times. In fact, from a given sequence, completion times are computed by scheduling tasks as soon as possible by following the sequence. The obtained schedule is then semi-active. Note that this property provides the basis for a lot of solving algorithms.

4.2. Existential property-based dominance rule

This dominance rule is expressed as "**there exists at least one optimal solution of S having the property A** ". The subset of solutions belonging to S satisfying A is also dominant. Note that an optimal solution may not satisfy A . From a practical point of view:

- Any solution which does not satisfy A can be removed from S , since there is at least one optimal solution of S satisfying A .
- Therefore, we can retain any identified subset of S containing the whole subset of solutions satisfying A , since this superset of A is dominant (see Proposition 1).

We refer to such a formulation as an **existential property-based dominance rule**. For example, "given an instance of $1|r_i|\sum T_i$, there is at least one optimal schedule which is semi-active" is such a rule [8]. Thus, only semi-active schedules can also be considered and it is also possible to use models in which only sequences are considered.

4.3. Sufficient property-based dominance rule

This dominance rule is expressed as "**any solution of S satisfying property A is optimal**": therefore A is a **sufficient condition of optimality according to S** . Note that an optimal solution may not satisfy A . From a practical point of view:

- Any solution which can be established satisfying A is an optimal solution of S .
- We can retain any subset of solutions for which this subset can be established it contains at least one solution satisfying A since such a subset is dominant (see Proposition 1). If such a subset does not exist or cannot be established, no adjustment can be performed.

We refer to such a formulation as a **sufficient property-based dominance rule**. For example, "given an instance of $1|\sum T_i$, any semi-active schedule in which tasks are sequenced in non decreasing order of processing times (*i.e.* the schedule is said to follow the shortest processing time order) and in which all tasks are tardy is optimal" is such a rule [26]. Of course, when building the semi-active schedule according to the shortest processing time order, there is at least one task which is on time, all semi-active schedules have potentially to be considered.

4.4. Necessary and sufficient property-based dominance rule

This dominance rule is expressed as "**all optimal solutions of S with respect to S have the property A , and any solution of S satisfying A is optimal**". Therefore A is a **necessary and sufficient condition of optimality with respect to S** . The subset of solutions belonging to S satisfying A is said to be strictly dominant. From a practical point of view:

- Any solution which does not satisfy A can be removed from S .
- Any solution which can be established satisfying A is an optimal solution of S .
- We can retain any identified subset of S containing the whole subset of solutions satisfying A , since such a subset is strictly dominant (see Proposition 1).
- We can retain any subset of solutions which can be shown to contain at least one solution satisfying A , since such a subset is dominant (see Proposition 1).

We refer to such a formulation as a **necessary and sufficient property-based dominance rule**. For example, "given an instance of $1|\sum C_i$, any semi-active schedule following the shortest processing time order is optimal" is such a rule [12]. This famous dominance rule, known as the Smith's rule, allows this problem to be solved in $O(n \log n)$ time.

4.5. Combining dominance rules

Emmons [26] pointed out that some dominance rules provide only existential properties, while others provide universal properties. From a practical point of view, this distinction is important. Indeed, universal property-based dominance rules can obviously be “accumulated”, while existential property-based dominance rules cannot. By “accumulation” Emmons means that if two valid universal property-based dominance rules take the form “all optimal solutions have the property A ” and “all optimal solutions have the property B ”, then clearly the dominance rule “all optimal solutions satisfy the properties A and B ” will hold.

It seems natural that existential property-based dominance rules cannot generally be directly “accumulated” in the above sense. Given two dominance rules δ_1 and δ_2 , set $\delta_1(S) \cap \delta_2(S) \cap \mathcal{O}(S)$ may be empty. Nevertheless, this does not mean that we cannot use different property-based dominance rules together if one of them is existential. In fact, dominance rule δ_2 should not be applied to S but to $\delta_1(S)$, taking into account the new constraints (implicit or explicit) which have been added by the application of the first dominance rule. Of course, the dominance checking involved by such an application may be really complex. Fortunately, the dominance rules described by Emmons for $1||\sum T_i$ can be accumulated, even though they are not universal. For example, one of these rules is expressed as “for any tasks i and k with $(k \notin B_i) \wedge (p_i < p_k \vee (p_i = p_k \wedge d_i \leq d_k)) \wedge (d_i \leq \max(\sum_{j \in B_k} p_j + p_k, d_k))$ then i precedes k ”, where B_x is the set of tasks which it has been decided will precede x using the same rule with other tasks. In this context, “ i precedes k ” means “there exists an optimal schedule in which all precedences previously established are satisfied and in which i is completed before k ”. Thus, while this dominance rule is an existential property-based dominance rule, it can be accumulated with itself (and two other rules of the same kind) if no inconsistencies are introduced in relation to the other pairs of tasks. To this end, sets B_x are used to be sure that the solution space will not be reduced to a set in which, for example, both $i \in B_k$ and $k \in B_i$. This kind of accumulation is common, particularly in enumerative methods where these kinds of added constraints are often implicit, *i.e.* taken into account in the variable domains.

5. Types of dominance rules

In this section we review the common characteristics of dominance rules encountered in the literature and we propose formal definitions of these characteristics.

5.1. Relation-based dominance rules

Dominance rules are often expressed as dominance relations. A formal definition of a dominance relation can be found in Kohler and Steiglitz [9, 25] and in Ibaraki [10] in the context of enumerative methods. Informally speaking, a dominance relation is a binary relation \succeq between two disjoint subsets of solutions S_1 and S_2 whereby the best solution belonging to S_2 is not better than the best solution of S_1 .

Definition 5. A **dominance relation** is a partial, reflexive and transitive binary relation \succeq defined on $\mathcal{P}(\Omega)$ such that, given two disjoint subsets S_1 and S_2 of Ω , $S_1 \succeq S_2$ if it can be established that $\Phi(S_1) \leq \Phi(S_2)$. We say that S_1 dominates S_2 or that S_1 is dominant over S_2 , and that S_2 is dominated by S_1 .

For example, “given an instance of $1|r_i|\sum C_i$, if there exist two tasks i and k such that $(p_i \geq p_k) \wedge (r_i + p_i \leq r_k + p_k)$ then the set of schedules S_i sequencing task i first dominates the set of schedules S_k sequencing task k first [27]” is such a relation. Indeed, it can be proved that the schedule minimizing the total completion time in S_i has a total completion time lower than or equal to the total completion time of a schedule minimizing the total completion time in S_k . Note that it does not mean that S_i contains an optimal solution of the problem. It means that S_i contains a schedule minimizing the total tardiness in $S_i \cup S_k$. Therefore, in the previous definition, a dominance relation can be seen as a special case of a dominance rule applied to the set $S_1 \cup S_2$ (or to any superset of $S_1 \cup S_2$) where the property identifying the dominant subset is the dominance relation.

Thus, when it is possible to find a relation whereby the best solution belonging to S_1 is strictly better than the best solution belonging to S_2 we shall refer to a strict dominance relation:

Definition 6. A *strict dominance relation* is a binary relation $>$ over $\mathcal{P}(\Omega)$ satisfying $S_1 > S_2$ implies $\Phi(S_1) < \Phi(S_2)$. We say that S_1 *dominates strictly* S_2 or that S_1 is *strictly dominant over* S_2 and that S_2 is *strictly dominated by* S_1 .

Note that special requirements must usually be met in order for dominance relations to be used correctly, according to the method used. In particular, a dominance relation is often required to be antisymmetric so as to be able to handle disjoint subsets S_1 and S_2 where $\Phi(S_1) = \Phi(S_2)$ (see for example [10]): at least one of the two subsets has to be retained.

5.2. Dominance rules for optimality and feasibility

Some dominance rules are established to identify subsets of solutions with at least one optimal solution, while others are established to eliminate unfeasible solutions. Whereas the former use the objective function to identify the dominated parts, the latter make use of the constraints of the problems. We can therefore establish the two following classes of dominance rules:

Definition 7. An *optimality-led dominance rule* is a dominance rule δ where the characteristics of the dominated part are expressed according to the objective function of the problem. Applied to a subset $S \subseteq \Omega$, the conditions of the dominance rule establish that $\Phi(\delta(S)) \leq \Phi(S - \delta(S))$.

Definition 8. A *feasibility-led dominance rule* is a dominance rule where the characteristics of the dominated part are expressed in terms of feasibility according to the constraints of the problem. Such a rule is commonly called an *elimination rule*. In a search problem, the dominated part should only contain unfeasible solutions.

For example, a feasibility-led dominance rule for $1|r_i, \bar{d}_i|\sum C_i$ (i.e. when deadlines are considered and the total completion time has to be minimized) could be “if there exist two tasks i and k such that $(r_k + p_k > \bar{d}_i - p_i)$ then the domain of $D(C_i)$ can be adjusted to $\{r_i + p_i, \dots, \min(\bar{d}_i, \bar{d}_k - p_k)\}$ and the domain of $D(C_k)$ to $\{\max(r_k + p_k, r_i + p_i + p_k, \dots, \bar{d}_k)\}$ ” because there is no feasible schedule in which task k is completed before task i . Note that in the case of a decision problem, some dominance rules include some feasible solutions in the dominated part. However, such a dominance rule belongs to the class of optimality-led dominance rules in which the objective function is two-valued (acceptable or unacceptable, see Section 2.5).

For example, it is the case when the previous dominance rule is applied to $|r_i, \bar{d}_i|_-$ in which the objective function is two-valued. Note also that a feasibility-led dominance rule is a redundant dominance rule (see Definition 4).

5.3. Morphism-based dominance rules

Some dominance rules depend on a transformation function over solutions. Informally, given two sets S and S' of solutions such that $S' \subset S \subseteq \Omega$, a morphism-based dominance rule establishes that if any solution belonging to S' can be transformed into a dominant solution belonging to $S - S'$, then S' is dominated (see Figure 4):

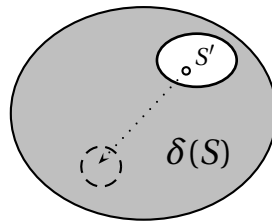


Figure 4: Morphism-based dominance rules.

Definition 9. Let S be a subset of Ω . Given a subset S' of S , a **morphism-based dominance rule** is a dominance rule depending on a morphism $\mu: \mathcal{P}(S') \rightarrow \mathcal{P}(S - S')$ preserving validity so that to any solution $z \in S'$ there corresponds a solution $\mu(z) \in S - S'$ such that $z \in \mathcal{F}(S) \Rightarrow \Phi(\mu(z)) \leq \Phi(z)$. The subset S' is then a dominated subset of S .

For example the dominance of semi-active schedules for the single-machine problem described in Section 3 depends on the application which associates with any non-semi-active schedule the only semi-active schedule following the same sequence of jobs obtained by removing all superfluous idle times. Commonly-encountered special cases of such dominance rules rely on isomorphisms that preserve the validity (feasibility or unfeasibility). Informally, an isomorphism-based dominance rule establishes that a subset of solutions S_2 is dominated as being isomorphic to another dominant subset of solutions S_1 (see Figure 5).

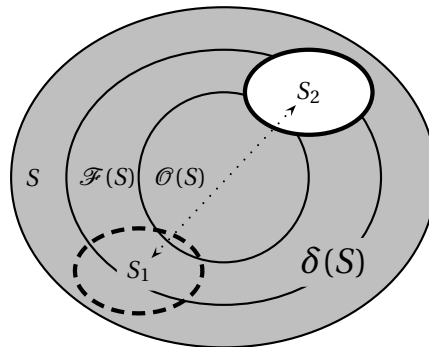


Figure 5: Isomorphism-based dominance rule.

Definition 10. Let S be a subset of Ω . Given two disjoint subsets S_1 and S_2 of set S , an **isomorphism-based dominance rule** is a dominance rule relying on a bijective mapping function $\mu : S_2 \rightarrow S_1$ preserving validity such that to any solution $z \in S_2$, there corresponds a solution $\mu(z) \in S_1$, and where it may be established that $\Phi(S_1) \leq \Phi(S_2)$. Subset S_2 is then a dominated subset of S . Subsets S_1 and S_2 are said to be isomorphic.

Note that while such a dominance rule will often satisfy $z \in \mathcal{F}(S) \Rightarrow \Phi(\mu(z)) \leq \Phi(z)$, the definition does not require this to be the case. Indeed, only the best solution belonging to S_1 (in the definition) needs to be at least as good as the best solution belonging to S_2 .

Such dominance rules were identified very early on as being an important factor in the efficiency of solution methods. Thus, in 1960, Swift wrote [14]: "*In many problems involving exhaustive searches the limiting factor with respect to speed or completion may not be the efficiency with which the search as such is conducted but rather the number of times the same basic problem is investigated. That is, the search routine may be effective in rejecting impossible cases in large blocks and still fails to accomplish its purpose in that the cases which must be investigated are looked at too frequently.*" In decision problems we often talk about symmetry breaking (see for example [28]). As far back as 1874 an article proposed making use of symmetries in the eight queens problem [29]. This subject has been addressed by a number of researchers in the past decades, particularly in the constraint programming community (see for example [28, 30, 31, 32, 33, 34, 35, 36, 37]). Note that there are several definitions of symmetry which differ in the attributes they apply to (only the values, only the variables, or variable-value pairs) and in what they preserve (the constraints or the set of solutions) [36].

5.4. Evaluation-based dominance rules

Some optimality-led dominance rules (see Section 5.2) rely on the use of a lower bounding function to identify dominated parts. This kind of dominance rule is notably the key element in the "bounding" operation for branch and bound methods. Whereas this operation is often described separately from the set of dominance rules, it can obviously be considered as a special case of dominance rules [10]. Consider a subset S of Ω . A function $lb : \mathcal{P}(\Omega) \rightarrow \mathbb{R}$ is a lower bounding function if it satisfies the following conditions [10]:

- $lb(S) \leq \Phi(S)$;
- $lb(S) = \Phi(S)$ if S is a singleton $\{s\}$;
- $lb(S) \leq lb(S')$ if $S' \subseteq S$.

A lower bound-based dominance rule is then established via the following proposition:

Proposition 2. Given a lowering function $lb : \mathcal{P}(\Omega) \rightarrow \mathbb{R}$, consider a solution z and a subset $S \subseteq \Omega$ with $z \notin S$. If $lb(S) \geq \Phi(z)$ then $\{z\} \succeq S$ and S is dominated. If $lb(S) > \Phi(z)$ then $\{z\} \succ S$ and S is strictly dominated.

In the same way, upper bound-based dominance rules can be established for maximization problems.

5.5. Dependent/independent-instance dominance rules

While some dominance rules can be expressed at a global level of the problem and are valid for any instance, others can be applied only if certain conditions are satisfied regarding the data. The first sort can be referred to as **independent-instance dominance rules**, in contrast to the second sort, which we may term **dependent-instance dominance rules**.

For example, the dominance of semi-active schedules for $1||\sum T_i$ described in Section 3 is independent of the instance to be solved, since it is valid for any instance, while the Emmons rule described in Section 4.5 depends on conditions which have to be fulfilled by a pair of tasks.

5.6. Shaving

In the context of job-shop scheduling, Carlier and Pinson [38], and Peridy [39] introduced the concept of “global operations”, in parallel with Martin and Shmoys who used the term “shaving”. This kind of rule is based on a refutation principle. Informally, global operations use a feasibility test to determine whether or not there exists a solution that satisfies a given property A . They are based on the following proposition:

Proposition 3. *Let S be a subset of Ω with $\mathcal{F}(S) \neq \emptyset$. Given a property A , if it can be established (by another rule) that none of the solutions satisfying A belongs to $\mathcal{F}(S)$, then the set of solutions not satisfying A is dominant.*

For example the feasibility-led dominance rule described in Section 5.2 for $1|r_i, \bar{d}_i|\sum C_i$ relies on the fact that if $(r_k + p_k > \bar{d}_i - p_i)$ and if k is completed before i then the schedule is not feasible according to the disjunctive constraint between i and k . Because these rules use other feasibility-led dominance rules, they are called “global”. Peridy [40] proposed a uniform framework to classify the dominance rules according to their level of use. Thus, feasibility-led dominance rules are said to be of level 1 while global operations using dominance rules of level 1 for their feasibility test are said to be of level 2.

The concept of shaving can obviously be extended to the optimality constraint:

Definition 11. *Let S be a subset of Ω with $\mathcal{F}(S) \neq \emptyset$. Given a property A , if it can be established (by another rule) that none of the solutions satisfying A belongs to $\mathcal{O}(S)$ then the set of solutions not satisfying A is dominant.*

6. Other dominances

While dominance rules apply to sets of solutions, the notion of dominance can be extended between dominance rules themselves, between instances of a problem and between problems.

6.1. Dominances between dominance rules or between dominance relations

In some cases, two dominance rules can be compared according to their potential for discarding solutions. Informally, we say that a dominance rule δ is stronger than a dominance rule δ' (δ dominates δ') if it eliminates the same solutions as δ' , with the expectation of other eliminations in addition.

Definition 12. *Consider two dominance rules δ and δ' . δ dominates δ' if $\forall S \subseteq \Omega, \delta(S) \subseteq \delta'(S)$.*

For example, a feasible schedule is said to be active if no job can be completed earlier without either delaying another job or violating a constraint [18]. It is well known that the subset of active schedules is dominant for $1|r_i|\sum C_i$ and $1|r_i|\sum T_i$ [15, 8]. Moreover, any active schedule is also semi-active. Since the set of active schedules for a given instance of a problem is included in the set of semi-active schedules, the dominance rule on active schedules dominates the dominance rule on semi-active schedules.

In the same way, this idea of potential can be expressed through dominance relations [10]:

Definition 13. *Given two dominance relations (strict or not) \triangleright and $\bar{\triangleright}$, \triangleright dominates $\bar{\triangleright}$ if $\forall S, S' \subseteq \Omega$ with $S \cap S' = \emptyset$ then $S \bar{\triangleright} S' \Rightarrow S \triangleright S'$.*

For example, the following slightly modified version of the dominance relation described in Section 5.1: “given an instance of $1|r_i|\sum C_i$, if there exist two tasks i and k such that $(p_i > p_k) \wedge (r_i + p_i < r_k + p_k)$, then the set of schedules S_i sequencing task i first dominates the set of schedules S_k sequencing task k first” is dominated by the dominance relation as stated in Section 5.1 because $(p_i > p_k) \wedge (r_i + p_i < r_k + p_k) \Rightarrow (p_i \geq p_k) \wedge (r_i + p_i \leq r_k + p_k)$.

Dominance between dominance rules determines whether it is sufficient to use only one rule instead of both dominance rules in a solution procedure. Dominance checking can be really time consuming, and it may be advantageous to know when using both dominance rules will not be useful. However, although it might appear obvious that a stronger dominance rule would contribute more than a weaker one to the effectiveness of a problem-solving algorithm, this is not always the case, as shown by Kohler and Steiglitz [9, 25] or Ibaraki [10] in branch and bound methods for which several counterexamples can be easily built. Nevertheless, Ibaraki provides several classes of branch and bound methods for which a stronger dominance relation always results in a more effective algorithm in terms of generated states. This monotonicity property for such classes of algorithms has encouraged practitioners to seek the strongest possible dominance rules for the problem to be solved, unless the computational time required for establishing dominance becomes predominant [10]. Indeed, a stronger dominance relation may take too much time to be applied.

6.2. Dominances between instances of problems

In some cases it is useful to express a dominance relation between two instances of the same problem. Informally, an instance I' dominates an instance I if an optimal solution of I is not better than an optimal solution of I' . A dominance relation between two instances can be defined as follows:

Definition 14. *Consider two instances I and I' of the same problem whose solution spaces are respectively Ω and Ω' . If it can be established that $\Phi(\Omega') \leq \Phi(\Omega)$ then I' dominates I . Moreover, if it can be established that $\Phi(\Omega') < \Phi(\Omega)$ then I' strictly dominates I .*

Such dominance rules can be used for preprocessing (see Section 7.4) or for establishing lower bounds and upper bounds of the problem. For example, in scheduling it is common to build an instance I' from an original instance I by splitting tasks into unary processing time jobs. In certain cases, the obtained instance I' dominates I and may possibly be solved easily (see for example [41]). Solving this new instance could then produce a lower bound of the value of an optimal solution of I .

Note that when an instance I can be split into subproblems, I' can then correspond to the consideration of several subproblems which can be solved independently.

6.3. Dominances between problems

The notion of dominance can also be extended between problems, as we now define:

Definition 15. Consider two problems Π_1 and Π_2 . If it can be established that for any instance I_2 of problem Π_2 , an instance I_1 of problem Π_1 can be built such that I_1 (strictly) dominates I_2 , then Π_1 (strictly) dominates Π_2 .

As for dominance rules between instances of the same problem, such dominance rules are often used for preprocessing (see Section 7.4) or for establishing lower bounds and upper bounds of the problem.

For example, consider the same problem as $1||\sum C_i$ but in which the preemption of tasks is allowed (the problem is denoted as $1|pmtn|\sum C_i$): this means that the execution of a task can be interrupted to process another task and be terminated later. In such a case, non-preemptive schedules (*i.e.* schedules in which no task is interrupted) are dominant [8] because any preemptive schedule can be transformed into a non-preemptive schedule (which is also a solution of the problem) with a total completion time lower than or equal to the initial completion time. Therefore, given an instance of $1|pmtn|\sum C_i$, if we consider exactly the same instance (here, the transformation is the identity) for $1||\sum C_i$, then the solution space to consider is reduced, while any solution to $1||\sum C_i$ is also a solution to $1|pmtn|\sum C_i$. On the other hand, consider the problem with release dates $1|r_i|\sum C_i$. The problem obtained by allowing preemption, *i.e.* $1|r_i, pmtn|\sum C_i$, dominates $1|r_i|\sum C_i$ because solving an instance of $1|r_i, pmtn|\sum C_i$ always gives a lower bound of $1|r_i|\sum C_i$ for the same instance. Moreover, while $1|r_i|\sum C_i$ is NP-Hard in the strong sense, $1|r_i, pmtn|\sum C_i$ is solvable in $O(n \log n)$ time.

7. Dominance rules for solving combinatorial optimization problems

In this section we describe the main applications of dominance rules in solving combinatorial optimization problems. As we shall see, they can be used at different stages of the problem solving process, and in the main general resolution methods.

7.1. Dominance rules and problem modeling

As stated in Section 2.1, modeling is an important stage where the variables and constraints of the problem are decided. Even at this stage it is possible to make use of dominance rules. Indeed, if we can consider a subset of solutions containing at least one optimal solution (or all optimal solutions), and if this subset of solutions can lead to a special model in which only these solutions are considered, the solution space considered by the solving algorithm is of course smaller, leading to a more effective method.

For example, consider the *eight queens problem*. At the modeling stage we use a dominance rule stipulating that *in all optimal solutions, there is exactly one queen per column* (see for example [20]). We then obtain a model with only eight variables $\{X_1, \dots, X_8\}$, where each variable X_i corresponds to the line of the queen located in column i . It is easy to see that this model leads to a solution space considerably smaller than when two variables are considered in order to locate each queen on the chessboard.

7.2. Dominance rules and the building of solutions

Some algorithms directly build solutions by taking into account a dominant subset of solutions. While the dominance rule is not used in the structure of the model, as it was in the previous section, it manifests itself in the way the solution is built by a solving algorithm.

For example, we have seen in Section 4 that the set of semi-active schedules is dominant for $1|r_i|\sum C_i$ and $1|r_i|\sum T_i$ allowing solving algorithms for these problems to build sequences of tasks instead of using starting times of tasks as decision variables. The starting times are obtained by scheduling the jobs as early as possible in the order of the sequence (see for example [7]).

7.3. Dominance rules as decision rules

Some algorithms are designed to deal with a specific problem, and most of them rely on dominance properties. They exploit interesting properties of the solutions being sought, and often belong to the class of polynomially solvable problems [17]. These properties are usually sufficiently strong to specify the structure of the optimal solutions which can then directly be built. Since the properties used are specific to the given problem, most of the time these algorithms cannot be used to solve other problems.

For example, Smith's rule described in Section 4.4 for the solving of $1||\sum C_i$, is such a decision rule. An instance of this problem can be solved simply by iteratively choosing the task with the shortest processing time and by scheduling it as soon as possible. Thus, in scheduling, decision rules are often described by giving a criterion (the processing time of the task in the example) which allows the next job to schedule to be identified. These criteria are commonly called **priority rules**.

7.4. Dominance rules and preprocessing

Preprocessing is a phase that occurs after the problem has been modeled but before it has been solved. It consists of operations that can be performed to improve or simplify the formulation by tightening bounds on variables, fixing values, *etc.* [19]. The aim is to prepare the formulation for the chosen solution method. The use of dominance rules is natural at this stage in the solution process. Different operations can be performed according to dominance rules:

- They facilitate the assignment of fixed values to some variables which can then be eliminated from the model.
- They may allow new constraints to be added, thus reducing the search space.
- They may allow the bounds of the variables to be tightened.
- They facilitate the splitting of a problem into subproblems.

7.5. Dominance rules in evaluation methods

When there is no possibility of an optimal solution (because of the size or of the intractability of the problem), or no requirement for one, suboptimal solutions may only be sought using approximate algorithms. Approximate methods include any way that can provide a "good" or near-optimal feasible solution with low computational requirements. Among these, greedy algorithms with their more or less complex variants and metaheuristics [42] appear to be very

effective for a wide variety of problems. Thus, in minimization (respectively maximization) methods, approximate algorithms produce upper bounds (respectively lower bounds) of the value of the objective function. Lower bounds (respectively upper bounds) of problems are important information in minimization (respectively maximization) problems. Indeed, they provide indications about the quality of the solutions being built, and they can improve the behavior of solving algorithms, particularly in enumerative methods (see Section 5.4).

Both in approximate algorithms and in the production of bounds, dominance rules are of significant interest. A lot of heuristics use dominance rules to improve the quality of the solutions built (see for example [43, 44, 45]). Dominances between instances of problems or between problems are particularly useful for the development of bounds (see Sections 6.2 and 6.3).

7.6. Dominance rules and linear programming

Many problems belong to the class of problems solvable with **linear programming**. These problems are characterized by a linear objective function, linear inequality constraints and variables whose domain is the set of real numbers. The solution space defined by the linear inequality constraints is a convex polytope. These problems can be solved using the “simplex algorithm” [46] which proceeds from one vertex of the polytope to another one by improving each time the value of the objective function. Dominance rules can be exploited as preprocessing to add some inequalities during a preprocessing phase (see Section 7.4). One way to solve a combinatorial search problem is also to identify a dominant set of solutions which can be expressed across a linear programming model.

In **integer linear programming** methods (see for example [19]), the problem is formulated with a set of linear inequalities which describe a convex polyhedron enclosing points corresponding to the subset of feasible solutions of the combinatorial problem whose coordinates are integers. A variant of the simplex algorithm can then be applied in which additional inequality constraints, called “**cutting planes**”, are generated as needed as the computation progresses. Cutting planes limit the solution space and can be considered as feasibility-led dominance rules identifying a dominant subset of solutions.

7.7. Dominance rules and dynamic programming

A large number of combinatorial problems (see for example [47, 48]) fit into the recursive framework of **dynamic programming**: solving a problem of order N requires solving subproblems of order $N - 1, N - 2, \dots, 1$, where the solution of a subproblem of order i can be expressed in terms of solutions of order $i - 1$. This allows a recursive approach for solving the original problem from the solutions of the subproblems.

This approach is based on Bellman’s principle of optimality [49] which can itself be considered as a dominance rule: “*an optimal sequence of decisions (a policy) has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions*”. Thus, an optimal solution can be built from locally optimal solutions of subproblems. Using dynamic programming to solve a problem essentially means deciding how to separate the problem into subproblems in order to satisfy the principle of optimality. Such a property has to be found in the structural properties of the problem solutions, and entails the use of dominance rules. Moreover, dominance rules can be used to eliminate non-interesting solutions of subproblems during the execution of the algorithm.

Kohler and Steiglitz [25] establish the relationship between enumerative and dynamic programming algorithms by showing that a dynamic programming method is equivalent to an enumerative algorithm (see the following section) in which dominance relations are used. In fact, solutions generated by the dynamic programming method for the subproblem of order i are the non-dominated solutions according to the dominance relation used in the enumerative method. The states generated for both methods are then the same.

7.8. Dominance rules and enumerative methods

Among the most commonly-used methods for solving combinatorial problems are the **enumerative techniques** [50] that we encounter in the literature under a variety of names, such as “Backtrack Programming” [20]. These are used for solving problems for which there is no known way of avoiding the huge size of their solution space. The idea is to find a way of organizing an enumerated list of solutions, using as much information as possible, so as to eliminate intelligently **dominated parts** of the solution space as the computation progresses, without having to enumerate these dominated parts [25] explicitly.

Thus, the term “**partial enumeration**” is often used, meaning a procedure for systematically enumerating parts of the solutions of Ω and examining them in such a way as to ensure that, by enumerating a reasonable small number of solutions, we have implicitly examined all elements of the solution space [51], the ones which are not explored being dominated. Thus, the utility of the method comes from the fact that only a small part of the solution space actually needs to be enumerated [52]. The remaining solutions are eliminated from consideration through the application of dominance rules that establish that such solutions cannot be optimal or better than the best one already known.

Among these methods, the “**branch and bound**” procedure has been popularized by Land and Doig [53] to solve Integer Linear Programming problems, by Little *et al.* [54] to solve the traveling salesman problem, and by Ignall and Schrage [3] to solve scheduling flowshop problems. The branch and bound method has been examined and generalized by a lot of authors (see for example [55, 21, 56, 57, 58]). A formal description with a theoretical behavior study of branch and bound methods is provided by Kohler and Steiglitz [9, 25] in the context of permutation problems. The name “branch and bound” arises from two basic operations [52]:

1. **branching**, which consists in dividing a subset of solutions into smaller subsets;
2. **bounding**, which consists in establishing bounds on the value of the objective function over the subsets of solutions, so as to be able to use it with an evaluation-based dominance rule (see Section 5.4).

The branch and bound method needs a recursive application of the branching and the bounding operations. A “lower bound” on the value of the objective function for all solutions belonging to one subset of solutions means that we can stop the branching of this subset since it is established that no better solution than the one already known can be reached from this subset. The popularity of this rather general method comes from its simplicity and its ability to solve combinatorial problems even when the objective and the constraints are nonlinear, discontinuous or even non-mathematically defined [21]. In order to use the branch and bound method, it is only necessary to be able to describe the problem as a tree, in which each node represents a partial solution. In addition, it must be possible to fix, at each node, a lower bound on the objective function for all nodes that emanate from it [3]. Moreover, if dominance rules can be defined for identifying and eliminating active nodes leading to dominated solutions (because they are proved to be either not optimal or equivalent to other solutions

enumerated elsewhere in the search tree) this can dramatically improve the behavior of the branch and bound procedure [57]. However, this technique may require a good deal of judgment to be able to choose a lower bound, the potential use of an initial solution, the incorporation of complicated dominance checks, and the specification of a branching mechanism [9, 25].

Among the most recent techniques which can be considered as belonging (at least in part) to enumerative methods is the programming paradigm known as **constraint programming** (see for example [23, 22]). The power of the constraint programming method lies mainly in the fact that constraints of the problem are used in an active process termed “constraint propagation”, where certain deductions are performed in order to reduce the domain of the variables, thus reducing the computational effort. Constraint propagation removes values from the domains, deduces new constraints, and detects inconsistencies. Since constraint propagation cannot usually detect all inconsistencies, an enumerative method is then used to branch on the remaining possible decisions. Constraint propagation algorithms mainly rely on the exploitation of feasibility-led dominance rules (see Section 5.2). Isomorphism-based dominance rules (see Section 5.3) are also frequently used. Less often, optimality-led dominance rules are used (see for example [35, 59]). In constraint programming, there are three main ways of introducing dominance rules [33]:

- remodeling the problem according to dominance rules;
- adding statically (implicit or explicit) constraints to the model which will only be satisfied by solutions of dominant parts;
- adding dynamically constraints during the search to ensure that any solution which is dominated by a solution already considered will not be considered.

8. Conclusion

Many of the “problems of everyday life” are computable and can be handled using computers. Optimizing energy and raw material consumption and reducing costs have recently become crucial objectives, both for ecological and economic reasons. A lot of the problems in question can be viewed as combinatorial optimization problems, defined by a set of variables subject to constraints. The aim is to find an assignment of the variables which satisfies all the constraints and which maximizes or minimizes an objective function representing a performance measure of the solution. There are a wide range of applications, including the distribution of goods, production scheduling, planning, budgeting, facility location and transportation network design. Dealing with these problems is a challenge. Indeed, most of them belong to the class of NP-Hard problems. Most of the time, and even for small-scale problems, the number of possible assignments is so huge that it is not feasible to consider all possibilities. Finding ways to reduce the search spaces is therefore a key issue.

Dominance rules, which eliminate uninteresting assignments or select interesting ones, are important stratagems. The interest of dominance rules is obvious since they reduce the search space of combinatorial problems which are in the process of being solved. Informally, a dominance rule identifies, generally via the formulation of properties, a subset of assignments containing at least one optimal solution. Despite their extensive use in solving combinatorial optimization problems, no detailed description of them is, to our knowledge, to be found in the literature. Thus, in this paper we have formally defined dominance rules. We have

established how dominance rules can be formulated and characterized. Moreover, we have shown how useful dominance rules can be in solving combinatorial problems.

It should be noted that the elaboration of the model remains the most crucial step in the problem solving process. Dominance rules can be used at this stage to reduce the search space at the outset, as this paper has pointed out. However, the richness of the model and its associated expressive power are also highly significant. Using dominance rules in filtering methods clearly shows that the different structural properties of interesting solutions are revealed only in the nuances of an elaborated model. This is why we think that calling traditional models into question might be the first step towards discovering new solutions to optimization combinatorial problems.

References

- [1] A. Manne, Programming of economic lot sizes, *Management Science* 4 (2) (1958) 115–135.
- [2] R. Levitan, A note on Professor Manne's "dominance" theorem, *Management Science* 5 (3) (1959) 332–334.
- [3] E. Ignall, L. Schrage, Applications of the branch-and-bound technique to some flow-shop scheduling problems, *Operations Research* 13 (1965) 400–412.
- [4] B. Fox, Discrete optimization via marginal analysis, *Management Science* 13 (3) (1966) 210–216.
- [5] F. Proshan, T. Bray, Optimum redundancy under multiple constraints, *Operations Research* 13 (1965) 800–814.
- [6] H. Weingartner, D. Ness, Methods for the solution of the multidimensional 0/1 knapsack problem, *Operations Research* 15 (1) (1967) 83–103.
- [7] S. Elmaghraby, The one machine sequencing problem with delay costs, *Journal of Industrial Engineering* 19 (1968) 105–108.
- [8] K. Baker, *Introduction to sequencing and scheduling*, John Wiley and Sons, 1974.
- [9] W. Kohler, K. Steiglitz, Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems, *Journal of the ACM* 21 (1) (1974) 140–156.
- [10] T. Ibaraki, The power of dominance relations in branch-and-bound algorithm, *Journal of the Association for Computing Machinery* 24 (2) (1977) 264–279.
- [11] S. Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistic Quarterly* 1 (1) (1954) 61–68.
- [12] W. Smith, Various optimizers for single stage production, *Naval Research Logistics Quarterly* 3 (1956) 59–66.
- [13] R. Gomory, Outline of an algorithm for integer solutions to linear programs, *Bulletin of the American Mathematical Society* 64 275–278.
- [14] J. Swift, Isomorph rejection in exhaustive search techniques, in: *American Mathematical Society Symposia in Applied Mathematics*, no. 10, 1960, pp. 195–200.
- [15] B. Giffler, G. Thompson, Algorithms for solving production scheduling problems, *Operations Research* 8 (1960) 487–503.
- [16] K. Baker, D. Trietsch, *Principles of sequencing and scheduling*, John Wiley and Sons, 2009.
- [17] M. Garey, D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman, 1979.
- [18] P. Brucker, *Scheduling Algorithms*, Springer Lehrbuch, 1995.
- [19] G. Nemhauser, L. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1998.
- [20] S. Golomb, D. Baumert, Backtrack programming, *Journal of the Association for Computing Machinery* 12 (1965) 516–524.
- [21] N. Agin, Optimum seeking with branch and bound, *Management Science* 13 (4) (1966) 176–185.
- [22] F. Rossi, P. Van Beek, T. Walsh (Eds.), *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., 2006.
- [23] P. Baptiste, C. Le Pape, W. Nuijten, *Constraint-based scheduling, applying constraint programming to scheduling problems*, Vol. 39 of *International Series in Operations Research and Management Science*, Kluwer, 2001.
- [24] J. Carlier, P. Chrétienne, *Problèmes d'ordonnement: modélisation / complexité / algorithmes*, Masson, 1988.
- [25] W. Kohler, K. Steiglitz, *Computer and job-shop scheduling theory*, John Wiley & Sons, ed. E.G. Coffman, 1976, Ch. Enumerative and iterative computational approaches, pp. 229–287.
- [26] H. Emmons, One-machine sequencing to minimize certain functions of job tardiness, *Operations Research* 17 (1969) 701–715.
- [27] M. Dessouky, D. Deogun, Sequencing jobs with unequal ready times to minimize mean flow time, *SIAM Journal on Computing* 10 (1981) 192–202.

- [28] C. Brown, L. Finkelstein, P. Purdom, Applied algebra, algebraic algorithms and error-correcting codes, 1988, Ch. Backtrack searching in the presence of symmetry.
- [29] J. Glaisher, On the problem of the eight queens, Philosophical magazine series 4 48 (1874) 457–467.
- [30] E. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: proceedings of AAAI-91: the 9th National Conference on Artificial Intelligence, Anaheim, California, 1991.
- [31] J. Puget, On the satisfiability of symmetrical constrained satisfaction problems, in: Proceedings of ISMIS'93, LNAI 689, 1993, pp. 350–361.
- [32] B. Benhamou, Study of symmetry in constraint satisfaction problems, in: proceedings of PPCP'94: the 2nd Workshop on Principles and Practice of Constraint Programming, 1994, pp. 281–294.
- [33] B. Smith, Reducing symmetry in a combinatorial design problem, in: Proceedings of CP-AI-OR'01, the 3rd International Workshop on Integration of AI and OR techniques in CP, 2001.
- [34] T. Fahle, S. Schamberger, M. Sellmann, Symmetry in constraint optimization, in: proceedings of CP'2001: the 7th International Conference on Principles and Practice of Constraint Programming, 2001, pp. 93–107.
- [35] S. Prestwich, C. Beck, Exploiting dominance in three symmetric problems, in: SymCon'04: the 4th International Workshop on Symmetry in Constraint Satisfaction Problems, a Satellite Workshop of CP2004: the 10th International Conference on Principles and Practice of Constraint Programming, 2004, pp. 63–70.
- [36] D. Cohen, P. Jeavons, C. Jefferson, K. Petrie, B. Smith, Symmetry definitions for constraint satisfaction problems, in: Proceedings of CP2005, The 11th International Conference on Principles and Practice of Constraint Programming, 2005, pp. 17–31.
- [37] T. Walsh, General symmetry breaking constraints, in: proceedings of CP-2006: the 12th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, 2006.
- [38] J. Carlier, E. Pinson, Adjustments of heads and tails for the job-shop problem, European Journal of Operational Research 78 (1994) 146–161.
- [39] L. Peridy, Le problème de job-shop : arbitrages et ajustements, Ph.D. thesis, Université de Technologie de Compiègne, France (1996).
- [40] L. Peridy, Règles d'éliminations pour les problèmes d'ordonnement disjonctif, Habilitation à diriger des recherches, Université de Technologie de Compiègne, France (2007).
- [41] H. Belouadah, M. Posner, C. Potts, Scheduling with release dates on a single machine to minimize total weighted completion time, Discrete Applied Mathematics 36 (1992) 213–231.
- [42] F. Glover, G. Kochenberger, Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research and Management Science, Springer, 2003.
- [43] C. Chu, M.-C. Portmann, Some new efficient methods to solve the $n|1|r_i|\sum T_i$, European Journal of Operational Research 58 (1991) 404–413.
- [44] C. Chu, Efficient heuristics to minimize total flow time with release dates, Operations Research Letters 12 (1992) 321–330.
- [45] A. Jouglet, D. Savourey, J. Carlier, P. Baptiste, Dominance-based heuristics for one-machine total cost scheduling problems, European Journal of Operational Research 184 (3) (2008) 879–899.
- [46] G. Dantzig, Linear Programming and Extensions, Princeton, NJ: Princeton University Press, 1963.
- [47] M. Held, R. Karp, A dynamic programming approach to sequencing problems, SIAM Journal on Applied Mathematics 10 (1) (1962) 196–210.
- [48] E. Lawler, J. Moore, A functional equation and its application to resource allocation and sequencing problems, Management Science 16 (1969) 77–84.
- [49] R. Bellman, The theory of dynamic programming, Bulletin of the American Mathematical Society 60 (1954) 503–515.
- [50] R. Walker, An enumerative technique for a class of combinatorial problems, in: American Mathematical Society Symposia in Applied Mathematics, no. 10, 1960, pp. 91–94.
- [51] E. Balas, Discrete programming by the filter method, Operations Research 15 (1967) 915–957.
- [52] L. Mitten, Branch-and-bound methods: general formulation and properties, Operations Research 18 (1) (1970) 24–34.
- [53] A. Land, A. Doig, An automatic method for solving discrete programming problems, Econometrica 28 (1960) 497–520.
- [54] J. Little, K. Murty, D. Sweeney, C. Karel, An algorithm for the traveling salesman problem, Operations Research 11 (1963) 972–989.
- [55] P. Bertier, B. Roy, Procédure de résolution pour une classe de problèmes pouvant avoir un caractère combinatoire, Cahiers du Centre d'Études de Recherche Opérationnelle 6 (1964) 202–208.
- [56] E. Lawler, D. Wood, Branch-and-bound methods: a survey, Operations Research 14 (4) (1966) 699–719.
- [57] E. Balas, A note on the branch-and-bound principle, Operations Research 16 (1968) 442–445.
- [58] B. Roy, Procédure d'exploration par séparation et évaluation, Revue Française d'Informatique et de Recherche Opérationnelle 6 (1969) 61–90.

- [59] T. Walsh, Symmetry in constraint optimization, in: proceedings of SymCon'07, The Seventh International Workshop on Symmetry and Constraint Satisfaction Problems, 2007.