Ann Oper Res (2006) 147:143–174
DOI 10.1007/s10479-006-0064-1

MCS—A new algorithm for multicriteria optimisation in constraint programming

F. Le Huédé · M. Grabisch · C. Labreuche · P. Savéant

Published online: 22 August 2006

Abstract In this paper we propose a new algorithm called MCS for the search for solutions to multicriteria combinatorial optimisation problems. To quickly produce a solution that offers a good trade-off between criteria, the MCS algorithm alternates several Branch & Bound searches following diversified search strategies. It is implemented in CP in a dedicated framework and can be specialised for either complete or partial search.

Keywords Multicriteria optimization · Multicriteria decision making · Constraint programming

1. Introduction

The practice and development of Operations Research and Artificial Intelligence have shown that, in many cases, two complex issues have to be taken into account when addressing real-life optimisation problems. First, industrial problems are often highly combinatorial and it is often difficult to solve them fast. Secondly, the preference relation between solutions to a problem is generally complex and depends on several conflicting criteria.

On the one hand, Constraint Programming (CP) has been designed to solve combinatorial problems and successfully applied to large instances. It follows the Branch & Bound

F. Le Huédé (✉) · C. Labreuche · P. Savéant
THALES Research and Technology France, domaine de Corbeville, 91401 Orsay cedex
e-mails: fabien.lehuede@thalesgroup.com

C. Labreuche
e-mail: christophe.labreuche@thalesgroup.com

P. Savéant
e-mail: pierre.saveant@thalesgroup.com

M. Grabisch
LIP 6, Université Pierre et Marie Curie (UPMC), 8, rue du Capitaine Scott 75015 Paris
Université Paris I Panthéon-Sorbonne
e-mail: Michel.Grabisch@lip6.fr

principles and uses constraint propagation to efficiently explore the solution space and find an optimal solution. Unfortunately, complex subjective preferences are not easy to model in CP and CP solvers do not use any dedicated method for solving problems with elaborate preference relations between solutions. However, in many optimisation problems such as scheduling, transportation, timetabling and planning problems, the best solutions are often those that offer a good compromise between economic, political, social and commercial considerations.

On the other hand, MultiCriteria Decision Making (MCDM) proposes several methodologies and tools that are able to provide an accurate model of the preferences of an expert. Research in this area focuses mainly on constructing appropriate models for preference representation, and on the methodological aspects to be considered when making a correct elicitation of subjective preferences. MCDM models evaluate or rank a given set of solutions (also called alternatives). However, when this set is implicitly described, the evaluation of each alternative can be difficult due to the complexity of the problem or the size of the solution set.

When simultaneously addressing the preference modelling and combinatorial aspects of multicriteria combinatorial optimisation problems, a first technique consists in returning several diversified non-dominated solutions, from which an expert will be able to choose the *best*. In an *automated* or *semi-automated* decision context, the intervention of a decision maker during the solving process is generally impossible or very limited. In most cases, the software has to return a single solution and more time has to be spent on building an accurate model of an expert's preferences beforehand. In this context, optimisation algorithms have to integrate the main characteristics of the established relation in order to be able to solve the problem within an acceptable time. Our overall objective is to combine the preference modelling capacities of MCDM methods with the power of CP search algorithms to solve multicriteria combinatorial optimisation problems under automatic decision requirements.

A first study allowed us to integrate an MCDM model in CP thanks to a global constraint that propagates a multicriteria aggregation function (Le Huédé et al., 2002). In this article, we address the problem of guiding a Branch & Bound algorithm towards good solutions, when the quality of a solution depends on several conflicting criteria. In CP, this guiding is carried out by a pair of heuristics that determines the selection order of decision variables and the separation process of the variables' domains. These two heuristics describe the search tree. We call "search strategy" the combination of these heuristics with an exploration strategy. When dealing with multiple conflicting criteria, the improvement of one criterion generally causes the degradation of another. Hence, a strategy that is good for optimising one criterion is often quite weak for optimising another. In addition, due to interaction and compensation phenomena, a good solution generally offers a trade-off between criteria. In this context, it is rarely possible to identify logical and systematic principles for the definition of a good Branch & Bound search strategy.

Although this study has been carried out in the context of CP, the problems that are addressed and the proposed algorithms can generally be applied to any Branch & Bound approach when used in a multicriteria optimisation context.

1.1. Related work

Multi-Objective optimisation problems have been extensively studied (e.g., see (Ehrgott and Gandibleux, 2002) for a detailed state of the art). Nevertheless, few techniques really integrate MCDM models in order to return a solution that is optimal with respect to a multicriteria

objective function. In addition, the latest surveys also show that the Branch & Bound technique is seldom used for solving multicriteria optimisation problems.

1.1.1. Multi-objective optimisation in Constraint Programming

In Gavanelli (2002), M. Gavanelli proposes an interesting implementation of the ε-constraint algorithm (Chankong and Haimes, 1983) in CP, using Point Quad-Trees for the filtering and the storage of Pareto-optimal solutions.

The PICPA algorithm (Barichard and Hao, 2003) is based on evolutionary concepts and interval constraint propagation to determine both a precise region of the objective space that bounds the Pareto-optimal front, and a set of approximate solutions that are well spread into this region.

In Junker (2002), the Preference Based Search (PBS) algorithm addresses multi-objective optimisation problems which lack a clear monotonic aggregation function. It searches either *balanced*, *extreme* or *Pareto-optimal* solutions in a constraint satisfaction problem. The preference formalism used in this framework allows the user to express that some criteria are more important than others or that compromises between criteria should be found by using the leximin approach. Given a set of such preferences between the criteria of the problem, a master algorithm selects some criteria that are used as optimisation objectives in a constraint-based solver. Following the ε-constraint method (Chankong and Haimes, 1983), the algorithm proceeds through successive mono-objective optimisations in order to explore the search space and find the set of non-dominated solutions (each mono-objective optimisation uses a dedicated labelling strategy). The PBS preference formalism is particularly suitable when the preference model has to be simple (such as for configuration problems on the Internet). In our context, the drawback of this method is that it does not allow modelling very precise preference relations. As a result, the algorithm returns sets of incomparable solutions and is not suitable for automated decision. However, this approach brings to the fore the principle that a search on a criterion has to be processed with a dedicated labelling strategy or, in more general terms, a dedicated search strategy.

A more pragmatic approach considers solving a scheduling problem with three objectives aggregated by a weighted sum (Focacci and Godard, 2002). This approach aims at finding good solutions within a limited time. It introduces a framework for the simultaneous setting of bounding constraints on the criteria and on the overall evaluation in order to speed up the discovery of good solutions in the Branch & Bound process. Different bounding constraints can be used (such as imposing an improvement of an objective or tolerating a degradation of 10% of this objective). The authors compare the use of bounding constraints in several algorithms. The best method iteratively runs two Branch & Bound processes; each process is given a limited time and focuses the search on a given criterion by means of various bounding constraints. This has been validated by experiments on scheduling problems. However, even if it can speed up some searches, adding bounds on the objectives of the problem is very likely to suppress solutions. Although the same search strategy is used for the different Branch & Bound algorithms, there is clearly a advantage in guiding the search in diversified parts of the tree to find a good solution to the whole problem more quickly.

1.2. Contribution of this paper

Due to the complexity of defining a good search strategy for multicriteria problems and considering related work on this subject, we propose a new framework called MCS (MultiCriteria Search). This is based on a generic algorithm that alternates searches following

various mono-criterion strategies to find solutions of increasing quality with respect to the multicriteria preference relation. One of the major innovations of this algorithm is the *criteria choice heuristic*. This mechanism determines which criterion it is most interesting to improve with respect to the last solution, the upper bounds of the criteria and the aggregation function. It allows the algorithm to use the appropriate strategy for guiding the search towards diversified solutions. In addition, according to Focacci and Godard (2002), *local constraints* can be specified to explicitly constrain the search in order to improve a given criterion. Finally, defining the *stopping condition* allows the user either to construct a partial search algorithm or to ensure optimality.

In the following section of this paper, we provide some important background information on MCDM and we summarise the main principles of the propagation of a multicriteria aggregation function in CP. The other sections present the MCS algorithm and the components of the framework. To use this framework, we propose several functions for building different instances of the generic algorithm. The notion of criterion choice heuristic is further developed in Section 5 and we give two interesting heuristics. Section 6 presents two kinds of local constraint and a stopping condition to ensure that an optimal solution is found. Finally, we present our latest work on the specialisation of MCS for partial search. Regarding validation, we experimented the different instances of the MCS framework on the examination timetabling problem.

2. Modelling multicriteria preferences

In this paper, we focus on the *Multi-Attribute Utility Theory* (MAUT) framework (Keeney and Raiffa, 1976). In this framework, the value of a solution is expressed through an overall evaluation, computed by an aggregation function according to its performances on the criteria. Since many functions can be used to aggregate a set of values, we introduce the properties of a multicriteria aggregation function that are the most desirable in order to give a good representation of multicriteria preferences. In addition, we introduce the Choquet integral aggregation function. This function is a very general aggregation function which can model a wide range of decisional behaviours. In particular, it is parameterised by a set function, called *fuzzy measure* or *capacity*, for which efficient elicitation processes have been created (Grabisch and Roubens, 2000; Labreuche and Grabisch, 2003). Finally, for the integration of an MCDM model in CP, we recall four conditions, previously introduced in Le Huédé et al. (2002). These conditions ensure the arc-consistency of a global constraint that models a multicriteria aggregation function.

2.1. Preference modelling in Multicriteria Decision Making

Multicriteria Decision Making (MCDM) models subjective preferences in order to automate the determination of a preferred solution out of a set of alternatives. Hence, solving a typical multicriteria decision problem consists in modelling the way an expert ranks a set of potential solutions, described by a set of *attributes* or *points of view*. To achieve this objective, the Multi-Attribute Utility Theory is mainly concerned with the construction of additive utility functions.

Let us denote by $\mathcal{N} = \{1, \ldots, n\}$ the set of criteria. We assume a set of *solutions* or *alternatives* $\mathcal{S}$ among which the decision maker must choose. Each solution is associated with a vector $a \in \Omega$ of which components $a_i \in \Omega_i, i \in \{1, \ldots, n\}$ represent the value of the solution for each point of view to be taken into account in the decision making process (we

may also abusively talk about "a solution $a \in \Omega$"). A component a_i is called the *attribute* of a solution. Typically, in a multi-objective optimisation context, each attribute would correspond to an objective function. According to these attributes, the modelling of the decision makers preferences $\succeq$ is achieved through an *overall utility function* $u : \Omega \to \mathbb{R}$ such that:

$$\forall a, b \in \Omega, a \succeq b \Leftrightarrow u(a) \geq u(b), \tag{1}$$

where $\succeq$ is a complete preorder.

Classically, this overall evaluation function is split into two parts (Keeney and Raiffa, 1976):

- The *utility functions*, denoted $u_1(a_1), \ldots, u_n(a_n)$, map each attribute to a single satisfaction scale $\mathcal{E} \in \mathbb{R}$. They model the *performance* of a solution on the criteria and ensure *commensurateness* between criteria, which is essential when several values have to be aggregated.
- The *aggregation function* aggregates the values returned by $u_1, \ldots, u_n$ and establishes the overall evaluation:

$$\forall a \in \Omega, u(a) = \mathcal{H}(u_1(a_1), \ldots, u_n(a_n))$$

where $u_i : \Omega_i \to \mathcal{E}$ and $\mathcal{H} : \mathcal{E}^n \to \mathcal{E}$. $u_i(a_i)$ is called the *utility* or *score* of the alternative a on the criterion i.

A common value for the satisfaction scale $\mathcal{E}$ is the [0, 1] interval. Establishing commensurateness between criteria allows us to work with comparable values. For example, this implies that we are able to express that a makespan of 20 days corresponds to the same level of satisfaction as a maximum tardiness of 3 days. Furthermore, when using compensatory aggregators (such as the weighted sum), it is important to work on a *scale of difference*. This means that the difference between two values on the same criterion has to make sense. For example, it means deciding if an increase of one day in the delay of completion of a schedule leads to the same decrease of satisfaction when this schedule is 15 days long or has been already delayed by 50 days.

To construct utility functions in the MAUT framework, we use the MACBETH methodology (Bana e Costa and Vansnick, 1994) (and its associated software), which is based on measurement theory. In particular, it takes into account the subjective character of the preferences that are modelled. To build a utility function, MACBETH asks the user to give some reference levels for a criterion and some general indications on the difference of satisfaction between values of this attribute.

2.2. Properties of a multicriteria aggregation function

Aggregation functions are used to make a synthesis of a set of elements. In decision aid, an aggregation function aggregates commensurate values which model criteria satisfaction levels. Additive aggregation functions, usually used in MAUT, suppose *preferential independence* between criteria, which is seldom the case in decision making. As a result, we propose using an enhanced version of this theory, with a more general aggregation function. Indeed, some essential requirements have to be met by the aggregation function $\mathcal{H}$. Among these important properties, which are detailed in Marichal (1998), we denote:

– *Monotonicity* (M): $\mathcal{H}$ should be an increasing function. That is to say:

$$x_i > x_i' \Rightarrow \mathcal{H}(x_1, \ldots, x_i, \ldots, x_n) \geq \mathcal{H}(x_1, \ldots, x_i', \ldots, x_n).$$

  Indeed, if one solution is better than another on at least one criterion and has equal performances on the other criteria, it cannot be of lesser quality than the other solution. This property is called *Strict Monotonicity* when the right side of the implication is a strict inequality.
– *Continuity* (C): $\mathcal{H}$ should be continuous with respect to its argument as the preferences of the expert generally evolve progressively.
– *Stability for positive linear transformation* ($\mathcal{H}(rx_1 + t, \ldots, rx_n + t) = rH(x_1, \ldots, x_n) + t$), with $r > 0, t \in \mathbb{R}$.
  This implies that the order obtained with the aggregation function is not modified when the same linear transformation is applied to utilities.

In addition, $\mathcal{H}$ has to be consistent with the preferences of the expert and should satisfy (1) when $\succeq$ is defined. Consequently, if we want to use the same parametric model in several applications, the aggregation function has to be very flexible. It has to be able to model the importance of a criterion, but also interaction and compensation effects between criteria. Furthermore, in decision aid, the "black box" approach cannot be accepted. Given a model, we need some information on the relative importance of the criteria or on interaction phenomena. We say that $\mathcal{H}$ has to be *semantically interpretable*.

2.3. The Choquet integral

One of the most commonly used aggregation functions is the weighted sum. However, using this operator supposes that each criterion acts independently on the quality of a solution. In multicriteria decision problems, some more general aggregation functions such as the *Choquet integral* (Choquet, 1953; Grabisch, 1996) are often necessary in order to take into account not only the importance of each criterion, but also interaction phenomena between the criteria. In order to generalise the weighted sum, (Sugeno, 1974) proposes assigning weights not only to each criterion separately, but also to any coalition of criteria. This weighting corresponds to a set function called "fuzzy measure".

Definition 1 (Fuzzy measure (Sugeno, 1974)). Let $\mathcal{P}(\mathcal{N})$ be the power set of $\mathcal{N}$. A fuzzy measure μ on $\mathcal{N}$ is a function $\mu : \mathcal{P}(\mathcal{N}) \to [0, 1]$, satisfying the following axioms.

(i) $\mu(\emptyset) = 0,\ \mu(\mathcal{N}) = 1.$
(ii) $A \subset B \subset \mathcal{N}$ implies $\mu(A) \leq \mu(B)$.

Here, $\mu(A)$ represents the degree of importance of the subset of criteria $A \subset \mathcal{N}$. In the MCDM methodology, the fuzzy measure is established in order to model the decision maker preferences (Grabisch and Roubens, 2000). Then, the mono-dimensional utilities $u_1, \ldots, u_n$ are aggregated with the Choquet integral to produce the overall evaluation of an alternative.

Definition 2 (The Choquet integral (Choquet, 1953)). Let μ be a fuzzy measure on $\mathcal{N}$, and $u = (u_1, \ldots, u_n) \in [0, 1]^n$. The Choquet integral of u with respect to μ is defined by:

$$C_\mu(u_1, \ldots, u_n) = \sum_{i=1}^{n} u_{(i)}\big[\mu\big(A_{(i)}\big) - \mu\big(A_{(i+1)}\big)\big], \tag{2}$$

where (i) indicate a permutation on $\mathcal{N}$such that $u_{(1)} \leq \cdots \leq u_{(n)}$, $A_{(i)} = \{(i), \ldots, (n)\}$ and $A_{(n+1)} = \emptyset$.

The Choquet integral is continuous, increasing, idempotent, stable for positive linear transformation and linear for a given order of its components. An axiomatisation of its use for preference modelling has been introduced by Marichal in Marichal (1998).

The combined use of the Choquet integral and fuzzy measures enables very precise preference models to be built. In particular, solutions that are not supported (i.e., Pareto optimal solutions that cannot be reached by a weighted sum, whatever the weights may be), can be found by the Choquet integral if they offer a better compromise than other solutions. Various decision making behaviours, such as tolerance and veto can also be modelled. The Choquet integral being very flexible, many classical aggregation functions, such as the weighted sum, the min, the max, or the ordered weighted sum (Grabisch, 1995) are particular cases of this function.

Figure 1 shows two representations of the Choquet integral on two criteria. The first curve represents a case where the interaction between the two criteria is positive (they are said to be *complementary*). It models a preference relation where a solution has to be good on both criteria to be considered good. On the contrary, the righthand curve models *substitutive* criteria (i.e., negative interaction). In this case, a solution is considered good by the expert as soon as it is good on one criterion. We can observe that the Choquet integral does not only model that either *balanced* or *extreme* solutions should be preferred. It also models compensation effects, which are omnipresent in MCDM problems.

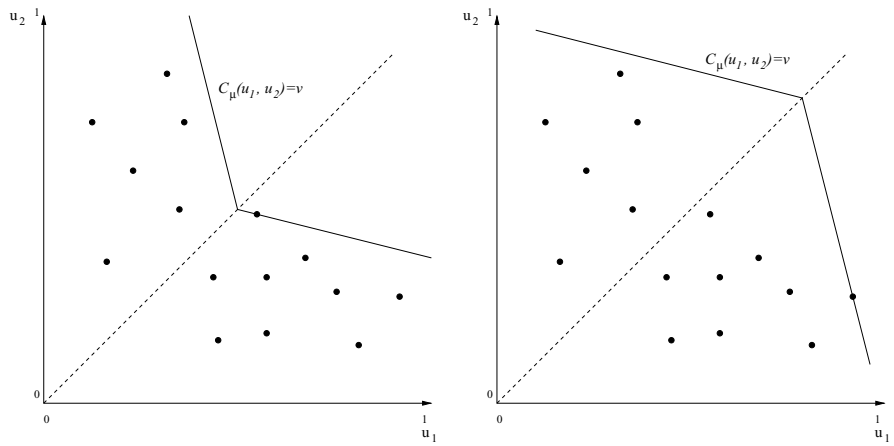


Fig. 1 Level curves of the Choquet integral for the aggregation of two criteria

Numerous practical applications and theoretical studies (Grabisch, Murofushi, and Sugeno, 2000; Marichal, 1998) have shown that fuzzy measures combined with the Choquet integral are particularly appropriate for aggregating utilities in a multicriteria decision problem. In addition, efficient tools and methodologies have been created in order to establish a good multicriteria model. In THALES, we use the MYRIAD© software to build the multicriteria model. MYRIAD allows the developer of the model to build the criteria hierarchy, call MACBETH to construct the utility functions (Section 2.1), and determine the Choquet integral coefficients according to the decision maker preferences. As these coefficients are complex, they cannot be set manually. Therefore, to calculate these values, the expert is asked to make several pairwise comparisons on various solutions. This constitutes a set of learning examples, used by MYRIAD to build a linear program and deduce the values of the fuzzy measure (Grabisch and Roubens, 2000).

2.4. Integration of a multicriteria aggregation function in CP

The integration of the Choquet integral in Constraint Programming has been presented in Le Huédé et al. (2002). The same principles can be used to integrate any multicriteria aggregation function (Le Huédé, 2003).

In summary, we consider n utility variables $u_1, \ldots, u_n \in [0, 1]$ that are connected with the attributes of the problem by the utility functions (modelled with piecewise linear constraints in our case). The global evaluation that will be optimised is modelled by the variable $y \in [0, 1]$. We aim to establish and propagate the equality between the y variable and the aggregation of $u_1, \ldots, u_n$ with a function $\mathcal{H}$. Mathematically, we want to enforce:

$$y = \mathcal{H}(u_1, \ldots, u_n)$$

Let us denote $Aggregation(\mathcal{H}, y, \{u_1, \ldots, u_n\})$ a global constraint that aims at enforcing this relation. The propagation of *Aggregation* can be achieved by maintaining the arc-B-consistency on this constraint. Let us denote $[\underline{x}, \overline{x}]$ the domain of a variable x.

Definition 3 (Arc-B-consistency (Lhomme, 1993)). Given a constraint c over q variables $x_1, \ldots, x_q$, and a domain $d_i = [\underline{x_i}, \overline{x_i}]$ for each variable x_i, c is said to be "arc-B-consistent" if and only if for any variable x_i and each of the bound values $v_i = \underline{x_i}$ and $v_i = \overline{x_i}$, there exist values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_q$ in $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_q$ such that $c(v_1, \ldots, v_q)$ holds.

Arc-B-consistency is weaker than the *arc-consistency* property. This is verified when, for each value in the domain of each variable, there is a set of values in the domain of the other variables that verifies the constraint.

If we suppose the monotonicity and the continuity of the function $\mathcal{H}$ applied on continuous variables, we can verify that an *Aggregation* constraint is arc-B-consistent by checking two conditions per variable:

Proposition 1 (Arc-B-consistency with respect to Aggregation). *Let $\mathcal{H}$ be a continuous and monotonous aggregation function and $C = Aggregation\ (\mathcal{H}, y, \{u_1, \ldots, u_n\})$ be an Aggregation constraint. C is Arc-B-consistent if and only if the following four conditions hold:*

(1) $\underline{y} \geq \mathcal{H}(\underline{u_1}, \ldots, \underline{u_n})$
(2) $\overline{y} \leq \mathcal{H}(\overline{u_1}, \ldots, \overline{u_n})$

(3) $\forall k \in \{1, \ldots, n\} \ : \ \mathcal{H}(\overline{u_1}, \ldots, \overline{u_{k-1}}, \underline{u_k}, \overline{u_{k+1}}, \ldots, \overline{u_n}) \geq \underline{y}$
(4) $\forall k \in \{1, \ldots, n\} \ : \ \mathcal{H}(\underline{u_1}, \ldots, \underline{u_{k-1}}, \overline{u_k}, \underline{u_{k+1}}, \ldots, \underline{u_n}) \leq \overline{y}$

Note that for such a continuous function on numeric variables, checking these four conditions also ensures that the *Aggregation* constraint is arc-consistent (Lhomme, 1993).

On a small example, maintaining the consistency of the Aggregation constraint gives the following propagations: Let y, u_1, u_2, u_3 be four variables and let $\mu = \{\mu_0, \mu_1, \ldots, \mu_{123}\}$ be a fuzzy measure such that: $y \in [0.4, 1]$, $u_1 \in [0, 0.2]$, $u_2 \in [0, 0.8]$, $u_3 \in [0, 0.2]$ and $\mu_0 = 0$, $\mu_1 = 0.1$, $\mu_2 = 0.4$, $\mu_3 = 0.1$, $\mu_{12} = 0.5$, $\mu_{13} = 0.2$, $\mu_{23} = 0.6$ and $\mu_{123} = 1$. If we set the constraint $Aggregation(C_\mu, y, \{u_1, u_2, u_3\})$, we obtain the following propagations: $C_\mu(0, 0, 0) = 0$ and $C_\mu(0.2, 0.8, 0.2) = 0.44$. Hence, conditions (1) and (2) allow us to deduce: $y \in [0.4, 0.44]$. Similarly, conditions (3) and (4) allow us to reduce the domains of u_1, u_2 and u_3: $u_1 \in [0.1, 0.2]$, $u_2 \in [0.7, 0.8]$ and $u_3 \in [0.12, 0.2]$ (see (Le Huédé et al., 2002) for a more detailed version of this example, and (Le Huédé, 2003) (in French) for the detailed propagation algorithms).

3. The MCS framework

The introduction of multicriteria aggregation functions in CP solvers generates another problem that is related to the CP solving process: on which principles should we base the construction of a Branch & Bound search when the objective function depends on several criteria?

In this section we investigate this issue further and we explain why it is nearly impossible to rationally parameterise a classical Branch & Bound method in multicriteria optimisation. Due to this major difficulty, we propose the MCS algorithm, which is based on the combination of several search strategies in a sequence of successive Branch and Bound searches.

3.1. Guiding the search in multicriteria optimisation with CP

In Constraint Programming, the search for solutions is generally guided by two processes that define a search tree. At a node of the tree, the first process, called *variable choice heuristic*, determines the next variable to be examined. The second process, called the *labelling strategy*, determines how the domain of the selected variable is split into several parts, and the order in which these parts will be considered. Generally speaking, these two processes are used together to build the search tree. An *exploration strategy* defines the order in which the nodes of this search tree are visited. In this paper, we will call *search strategy* the combination of a labelling strategy, a variable choice heuristic and an exploration strategy (the Depth-First Search (DFS) strategy being the default exploration strategy).

In optimisation, the objective of a search strategy is to quickly guide the search towards good solutions. This allows the algorithm to prune large parts of the search tree thanks to the propagation of the objective function. Therefore, a search strategy has to be defined according to the objective function. In CP as for any Branch & Bound method, the definition of a good search strategy has a great impact on the solving time.

Since MAUT models establish an overall evaluation for each solution, the optimisation problem we want to solve can be treated like any mono-objective problem. However, in this particular context, defining a good search strategy becomes much more difficult due to the multicriteria nature of the problem.

Indeed, the definition of a search strategy often relies on intuitive and logical principles (i.e. with respect to the properties of the problem). For example, when trying to minimise

the duration of exams in an examination timetabling problem, a good strategy is to select the examination to be taken by the largest number of students in the set of exams that are not yet scheduled, and to place it as early as possible in the timetable. If the objective is to minimise the number of rooms used, we will place the exams in the largest rooms first. However, when trying to find a good compromise between these two criteria, planning an exam in the biggest room can delay the end of the exams (there may be some smaller rooms available earlier), but it can also reduce the number of rooms used if this bigger room is able to host other exams that could not be contained simultaneously in a smaller room. Here, the logical principles contradict each other and the definition of a common logical rule is far from obvious.

Actually, two characteristics that occur very often in multicriteria problems prevent us from identifying a heuristic that would be efficient for all criteria simultaneously:

- Due to the constraints of the problem, criteria in the preference model often contradict each other (i.e., it is hard to improve one criterion without decreasing the satisfaction level of another).
- The expert is often looking for a good trade-off solution represented by compensation phenomena between the criteria in the preference relation (the decrease in satisfaction of one criterion can be accepted by the expert if it allows sufficient improvement on other criteria).

Hence, a strategy that is good for one criterion is very likely to be inefficient for another, and since good solutions are compromise solutions, heuristics have little chances of guiding the search towards them from the start. Consequently, a large part of the tree has to be visited before good solutions can be found.

Furthermore, many combinatorial optimisation problems are NP-hard, and therefore intractable, due to the size of a complete tree search. On such problems, only a limited amount of time (*time contract*) is often given for finding a solution that has to be as good as possible. Within this time limit, DFS generally performs poorly on large problems and partial search algorithms have been developed to improve the exploration of the search tree (Beldiceanu et al., 1997; Harvey and Ginsberg, 1995; Gomes, Selman, and Kautz, 1998). As a result, in the context of the design of a general algorithm for solving multicriteria optimisation problems in CP, we want to keep the capability of integrating partial search methods.

3.2. The MCS principles

According to the search problematic in a multicriteria optimisation context, we propose the MCS (MultiCriteria Search) algorithm that alternates diversified search strategies in order to allow good solutions to be found more quickly and reduce the average resolution time.

3.2.1. Diversification needs in multicriteria optimisation

A key point in the search for solutions in multicriteria optimisation is the exploration of solutions that are distant in the search space. This idea is related to the concept of *diversification* used in multi-objective meta-heuristics for the search of the Pareto-optimal front, and its advantage in CP has been shown in the related work introduced in Section 1.1.

In the MCS algorithm, we propose intensifying the search for diversified solutions thanks to the iterative use of several search strategies. This approach relies on the definition of a good search strategy per criterion, each strategy being used for the search of a solution that optimises its corresponding criterion. These mono-criterion searches are launched iteratively

to find some solutions of increasing quality with respect to the overall evaluation function. The hypothesis of our algorithm is that it is much easier to define a strategy per criterion than a global strategy and that the diversification brought by the alternation of these strategies should help to define better searches.

3.2.2. Algorithm overview

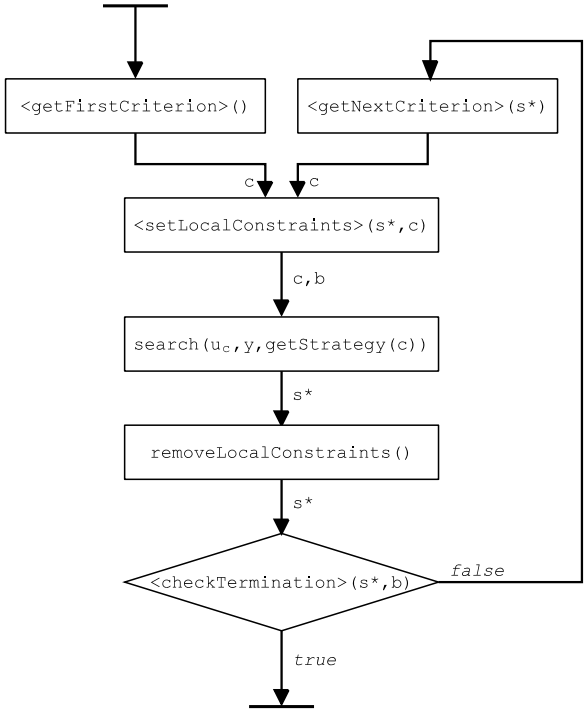
Figure 2 gives an overview of the MCS algorithm development.

As previously stated, this process relies on the definition of one search strategy per criterion. The main idea is to alternate different searches on the problem criteria and simultaneously to impose an improvement of the overall evaluation every time a solution is found.

The main strength of the MCS algorithm is that it proposes a dynamic selection of the search strategies, depending on the state of the search (i.e. on the result of the previous mono-criterion search). This selection is achieved by a function called the *criterion choice heuristic* which, before each search, chooses the criterion that will indicate the strategy to be used (we say that this criterion will "guide" the search). On the first iteration of the algorithm, a criterion is selected by a function denoted <**getFirstCriterion**>. Otherwise, this task is performed between each search by <**getNextCriterion**>, that will choose the criterion on which it is most interesting to get an improvement with respect to the last found solution, denoted $s^\star$.

In addition, before a mono-criterion search is launched, it is possible to explicitly post additional constraints in order to concentrate the search in interesting parts of the search space. These constraints, called *local constraints* and set in the <**setLocalConstraints**> component, are valid *only during the next mono-criterion search* (they are removed by the

Fig. 2 MCS flowchart



removeLocalConstraints() function). For example, they can impose an improvement on the selected criterion through bounding constraints as in Focacci and Godard (2002) or can be used to express more complex cuts.

The search is performed by the maximize function which searches for solutions that are better on both the selected criterion u_c and the overall evaluation y. The search strategy used during this search is given by getStrategy(c).

Finally, the end of the algorithm is triggered by the <**checkTermination**> component. This function is called the *stopping condition* and can use the result of the searches, a time contract, or any other user-defined information, according to the characteristics that are desired for the overall algorithm. For example, the stopping condition may stipulate that the algorithm stops when a mono-criterion search fails to find any solution.

3.2.3. MCS foundations

In a more formal way, the MCS algorithm is characterised by a sequence of dynamically chosen criteria that direct the mono-criterion searches by means of dedicated search strategies.

At each iteration of the algorithm, the inputs are: a MCDM problem $\mathcal{P}$, a current solution $s^\star$ (= nil initially), a current criterion c (= nil initially), a search strategy and a solution s, returned by the search performed by the CP solver at this iteration. Variables u_c and y model the evaluation of criterion c and the global evaluation respectively.

The search performed during this iteration is based on the following principles:

- The mono-criterion search is performed on a problem $\mathcal{P}'$. Before the search, $\mathcal{P}'$ is either equal to $\mathcal{P}$, or obtained from $\mathcal{P}$ by adding a *local* constraint $\mathcal{C}$ that reduces the search space for this MCS iteration.
- During the mono-criterion search on problem $\mathcal{P}'$, the criterion c and the overall evaluation y are optimised simultaneously. In other words, each time a solution s' is found, if we denote v' and v'' the respective values of u_c and y in s', constraints $(u_c > v')$ and $(y > v'')$ are added to $\mathcal{P}'$ and the search resumes as defined by the search strategy.
- If no solution is found during the search, $s = \text{nil}$.
- If $s \neq \text{nil}$ and v is the value of y in solution s, a new MCDM problem is obtained from $\mathcal{P}$ by adding the constraint $(y > v)$ and $s^\star$ is set to s.

In this paper we address the problem of finding an optimal solution as well as partial search with MCS. When optimality is desired, it is necessary to ensure that the solution found by maximisation of a given criterion on $\mathcal{P}'$ finally corresponds to an optimal solution of $\mathcal{P}$ after a finite number of MCS iterations.

As a result, some properties are needed to guarantee the correctness of the algorithm. We distinguish two kinds of properties: *Space Reduction Properties*, which determine when it is possible to add cuts in the search space without removing better solutions, and *Termination Properties*, which determine when the algorithm terminates.

Space Reduction Properties:

RP1 If s is a maximal solution for criterion c on the MCDM problem $\mathcal{P}$, v is the value of c in s, and $\mathcal{P}''$ is obtained from $\mathcal{P}$ by adding the constraint $(u_c \leq v)$, then the optimal solutions of $\mathcal{P}$ and $\mathcal{P}''$ coincide.

RP2 If v is the value of u_c in $s^\star$ and $\mathcal{P}'$ is obtained from $\mathcal{P}$ by adding the (local) constraint $(u_c > v)$, if there is no solution to the problem of maximising u_c on problem $\mathcal{P}'$, if $\mathcal{P}''$

is obtained from $\mathcal{P}$ by adding the constraint $(u_c \leq v)$, then the optimal solutions of $\mathcal{P}$ and $\mathcal{P}''$ coincide.

These properties are based on rather trivial optimisation principles but they can be useful to help the solver in reducing the search space during the solving process.

Termination Property:

TP1 If $\mathcal{P}' = \mathcal{P}$ and there is no solution to the problem of maximising u_c on problem $\mathcal{P}'$, then either $s^\star \neq \mathsf{nil}$ and $s^\star$ is optimal for y on the MCDM problem $\mathcal{P}$ (the constraint $(y > v)$, where v is the value of y in $s^\star$, has been added at the previous MCS iteration), or $s^\star = \mathsf{nil}$ and $\mathcal{P}$ is infeasible.

Hence, we can say that an optimal solution has been found when after several iterations of the algorithms, problem $\mathcal{P}$ becomes infeasible.

Of course, the characteristics of the final solution are closely connected to the properties of the functions that parameterise the MCS algorithm. In the following sections, we first give a more precise definition of the algorithm and of the components that characterise the search. A more formal characterisation of the solution returned by the MCS algorithm is then given in Section 6, whereas Section 7 describes a specialisation of the framework dedicated to partial search.

3.3. The MCS algorithm

From the algorithmic point of view, the MCS principles are expressed in Fig. 3:

Variables
In this algorithm, the criteria of the problem are modelled by the vector of variables $u = (u_1, \ldots, u_n)$. These variables are connected to a variable y by an Aggregation (Section 2.4)

```
MCS
    u ← (u_1, ..., u_n), s* ← nil, s ← nil, c ← 0, local ← false
    setConstraint(aggregation(H, y, u))
    while not(<checkTermination>(local, s*, s, y, u, c))
        if (c = 0)
            c ← <getFirstCriterion>(H, u)
        else
            c ← <getNextCriterion>(s*, H, u)
        endif
        local ← <setLocalConstraints>(s*, u, c)
        s ← maximize(u_c, getStrategy(c))
        removeLocalConstraint()
        if (s ≠ nil)
            s* ← s
        endif
    endwhile
    return s*
end
```

Fig. 3 The MCS algorithm

constraint to establish the overall evaluation. The variables s and $s^\star$ are used to store the solutions that are found by the algorithm and they are equal to nil when they do not contain any solution. Furthermore, at any time during the search, the best solution is stored in the variable $s^\star$, whereas s is used as an intermediate variable. As in the flowchart, the c variable models the index of the criterion that has been chosen by the criterion choice heuristic. Finally, the *local* boolean variable indicates if a local constraint has been set or not before a search.

Searches and constraints
The setConstraint function is used to post constraints to the problem. Function maximize optimises the criterion u_c following the strategy given by getStrategy(c). In parallel with this algorithm, constraint $y > y^\star$ is set every time a solution of overall satisfaction $y^\star$ is found. Thus, this "optimisation constraint" imposes that solutions of increasing quality must be found over the various searches, whereas the constraints used by maximize to optimise u_c are removed after every search.

Generic functions
In addition to defining strategies, we recall the three functions through which the algorithm can be customised

- **Criterion choice heuristic:** defined in the function <**getNextCriterion**>, it returns the index of the criterion that will guide the next search.
- **Local constraints:** depending on the chosen criterion they can be added before each search in the <**setLocalConstraints**> function. This function returns a boolean value to indicate if a local constraint was in fact set or not.
- **Stopping condition:** defined in the <**checkTermination**> function, the stopping condition of the algorithm is launched after each mono-criterion optimisation.

3.4. Definition of a generic search algorithm in the CP context

The problem of defining a search strategy is common to any multicriteria optimisation problem where criteria are aggregated as in MAUT, and where the solving process is based on the construction of a search tree. The objective of our research is however to provide a CP solver with search facilities for multicriteria problems. The main asset of this particular context is the supply of search definition facilities for classical optimisation, that can be easily reused for our purpose. However, the requirements are that the constructed algorithm should be easy to use and to adapt to many kind of problems. Thus, the MCS algorithm has been designed in a framework. For each component of the algorithm, the framework provides a library of functions that can be used in many problems or redefined for specific use.

The MCS algorithm has been implemented in the Eclair© solver PLATON TEAM (2001) and can be invoked by the following functional call: MCS($y, (u_1, \ldots, u_n), (S_1, \ldots, S_n)$, <getNextCriterion>, <setLocalConstraints>, <checkTermination>), where $(S_1, \ldots, S_n)$ are search strategies described using the ToOLS© library (de Givry and Jeannin, 2003). ToOLS is a framework that offers primitives to easily describe search trees. It allows the expression of most of the partial search methods, but also some hybrid local/global search methods such as Large Neighbourhood Search (Shaw, 1998).

In the following sections of this paper we give some definitions for the MCS components that are general enough to be used in a large number of problems. The combination of these functions and the general algorithm compose the MCS framework.

4. Defining a mono-criterion search strategy for the MCS framework

The first essential component of the MCS framework is the set of search strategies that will guide the mono-criterion optimisations. Naturally, as introduced in Section 3.1, the definition of a good variable choice heuristic and a good labelling strategy for each mono-criterion strategy is a critical point when using the MCS algorithm. In addition, many alternative tree searches have been developed in CP to perform a better exploration of the search space. This section discusses what should be considered when defining a set of search strategies for MCS. We first discuss the construction of the search trees and then, consider different possibilities for their exploration strategies.

4.1. Defining a good set of search strategies

When the criterion choice heuristic detects that it is interesting to get an improvement on a given criterion, the associated strategy should be able to quickly guide the search towards solutions of good quality on this criterion. Although this is the main requirement for a strategy, there can still be several good heuristics for this task. In mono-criterion optimisation, the choice of one heuristic is generally based on an empirical comparison of the different strategies. Hence, at first sight, a strategy that performs well in mono-criterion optimisation is a good candidate for use in MCS. However, it may also be important to compare the strategies chosen for each criterion and to prefer a set of strategies that offers a more diversified exploration of the search space.

Remark 1 (Back to diversification matters). The choice between two search strategies for a criterion in MCS can be made on the basis of mono-criterion optimisation benchmarks. However, one strategy can also be preferred to another in the MCS context if it is completely different from the strategy designed for the other criteria. Indeed, using diversified strategies can be a great advantage when the search gets closer to very good compromise solutions. In this case, it is often hard for the search to find solutions of increasing overall quality. Diversified strategies have less chances to get stuck on these difficulties and may be able to find better solutions more quickly.

4.2. Using incomplete searches

For a search on a criterion c in the MCS framework, it is not really important to prove that the solution found is optimal on c. Although this may allow the algorithm to decrease the upper bound of u_c (and by propagation, the upper bound of the overall evaluation y), it may take a long time and not be worthwhile.

For this reason, it is possible to set limits on the mono-criterion strategies in order to reduce the search time that is left to mono-criterion searches and favour the alternation of strategies. This idea is close to the equilibrium between *intensification* and *diversification* in multi-objective meta-heuristics.

Hence, at least two kinds of simple limits can be specified:

- **Limits on the number of solutions:** a mono-criterion search stops when it has found a given number of solutions.
- **Temporal limits:** a time contract is given to each mono-criterion search.

However the definition of limits such as time limits should be done carefully. Indeed, with such a limit, if a search does not find any solution, it does not mean that there is no solution to the problem.

More complicated limits can also be set to modify the exploration order of the nodes in the search tree. This allows us to use alternative exploration strategies and is often employed for partial search. Among the most popular partial search methods in CP, *iterative weakening* methods propose to iteratively solve the same problem with a limit on the development of the search tree, the limit being relaxed over the various iterations of the algorithm. The main objective of these methods is to backtrack earlier than the classical depth-first strategy, on the decision made on the top of the tree. Some example of such methods are *LDS* (Limited Discrepancy Search) (Harvey and Ginsberg, 1995), *Iterative Broadening* (Ginsberg and Harvey, 1992) and *Depth-bounded Discrepancy Search* (Walsh, 1997).

5. Choosing a criterion to guide the search

When a mono-criterion search ends and a solution has been found, the function <**getNext-Criterion**> determines the criterion that will guide the next search. A simple criterion choice heuristic optimises each criterion one after the other in a static given order. It allows MCS to explore diversified areas of the solutions space. However, this process can be considerably improved by dynamically choosing the search strategy that is most likely to find a large improvement of the overall evaluation.

At the end of a search, we can use the following information:

- The last found solution, represented by the $u^\star = (u_1^\star, \ldots, u_n^\star)$ vector, which contains its performances on the problem criteria (this solution is also called "current" solution).
- The bounds of the utility variables, denoted $\underline{u} = (\underline{u_1}, \ldots, \underline{u_n})$ and $\overline{u} = (\overline{u_1}, \ldots, \overline{u_n})$. At this point of the problem, they represent the largest possible debasement or improvement on the problem criteria.

We can also exploit the parameters of the aggregation function $\mathcal{H}$.

The dynamic choice of a promising criterion to "guide" the next search can be achieved using an indicator that depicts the *worth to improve* on a criterion. This indicator aims to establish a measure of the algorithm interest in using a strategy during the next search. Thus, a criterion choice heuristic can be based on such indicators if the heuristic returns the index of the criterion that has the highest value for a given *worth to improve indicator*.

In this section, we propose some properties for worth to improve indicators. We then define two useful indicators: the *maximum improvement indicator* and the *average improvement indicator*, and we apply them to the Choquet integral case.

In the following, given a vector of real values $(u_1^\star, \ldots, u_n^\star)$ and a real x, we denote $(u^\star_{-i}, x) = (u_1^\star, \ldots, u_{i-1}^\star, x, u_{i+1}^\star, \ldots, u_n^\star)$.

5.1. Properties of a "worth to improve" indicator

Let us denote $w_i(\mathcal{H})(u^\star, \overline{u})$ a function of $\mathcal{E}^n \times \mathcal{E}^n$ that takes its value in $\mathbb{R}$. The variables of this indicator are vector $u^\star$ of the last solution performances on the problem criteria, vector $\overline{u}$, that models the score upper bounds, and the aggregation function $\mathcal{H}$ that establishes the overall evaluation. The main desirable properties for this function are the following (Labreuche, 2004):

(G) *Gain:* if it is possible to improve a criterion, the "worth to improve" indicator should be positive:

$$\text{if } \overline{u_i} > u_i^\star \text{ then } w_i(\mathcal{H})(u^\star, \overline{u}) > 0$$

(NG) *No Gain:* if it is not possible to improve a criterion, then the "worth to improve" indicator for this criterion should be null:

$$\text{if } \overline{u_i} = u_i^\star \text{ then } w_i(\mathcal{H})(u^\star, \overline{u}) = 0$$

If these properties are verified we can denote that, during the search, either there exists a criterion that has a positive worth to improve indicator, or the last found solution is optimal ($u^\star = \overline{u}$).

5.2. The maximum improvement indicator

The maximum improvement indicator relies on the (ideal) hypothesis that if we use the strategy associated to a criterion i during the following search, we will find a solution that reaches the highest possible performance $\overline{u_i}$ on criterion i and that is equivalent to the current solution on the other criteria. This makes the hypothesis of a kind of functional independence between the criteria, which is not true due to the problem constraints. In consequence, the information returned by this indicator gives the main trends but is only heuristic.

The maximum improvement indicator for each criterion is denoted $\chi_i(\mathcal{H})$ $(u^\star, \overline{u})$ and is defined by:

$$\chi_i(\mathcal{H})(u^\star, \overline{u}) = \mathcal{H}(u^\star_{-i}, \overline{u_i}) - \mathcal{H}(u^\star) \tag{3}$$

Knowing the upper bound of the utility variable of criterion i and assuming that the other criteria values will not change, the $\chi_i(\mathcal{H})(u^\star, \overline{u})$ indicator represents the maximum value that can be reached by the overall evaluation if we try to improve the i^{th} criterion.

In the MCS framework, the criterion choice heuristic that is built on this indicator is called maxImprovement_h.

5.3. The average improvement indicator

The average improvement indicator, denoted $\omega_i(\mathcal{H})(u^\star, \overline{u})$, is a particular case of the ratio proposed by Grabisch and Labreuche (2001) and Labreuche (2004). This ratio was initially used to determine on which criteria an alternative (for example, a trainee), should improve first to get the highest increase of this overall evaluation.

The average improvement indicator is based on the same idea as the maximum improvement indicator but it takes into account the fact that the maximum utility value may not be reached on the chosen criterion. Therefore, it may be better to consider an average value of the improvement of the overall evaluation when the value of a criterion ranges between the last value found and its maximum value:

$$\omega_i(\mathcal{H})(u^\star, \overline{u}) = \begin{cases} 0 \text{ if } u_i^\star = \overline{u_i}, \\ \dfrac{1}{\overline{u_i} - u_i^\star} \displaystyle\int_{u_i^\star}^{\overline{u_i}} \left(\mathcal{H}(u^\star_{-i}, t) - \mathcal{H}(u^\star)\right) dt \text{ otherwise.} \end{cases} \tag{4}$$

In a probabilistic sense, under the hypothesis of a uniform law for the increases of u_i, this ratio can be considered as the expected value of the global improvement generated by a search on criterion i.

To facilitate implementation of the average improvement indicator applied to the Choquet integral, we propose adapting the formula of (4) when $\mathcal{H}$ is replaced by C_μ (where C_μ denotes the Choquet integral, as given in Definition 2). To achieve this, we use the Möbius transform, an alternative representation of the fuzzy measure (Grabisch, 1997), which is often employed for calculus convenience.

Definition 4 (*The Möbius transform).* Given a measure μ, the Möbius transform of μ is given by:

$$m(A) := \sum_{B \subset A} (-1)^{|A \setminus B|} \mu(B), \quad \forall A \subset X.$$

According to this representation, the formula of $\omega_i(C_\mu)$ can be expressed as follows:

Proposition 2. (Formula of the average improvement indicator applied to the Choquet integral). *Let m be the Möbius transform of μ (c.f. Grabisch (1997)). Then:*

$$\omega_i(C_\mu)(u^\star, \overline{u}) = \sum_{C \subset \mathcal{N}} m(C) \times \omega_i(h_C)(u^\star, \overline{u}) \tag{5}$$

with $h_C(u) = \bigwedge_{j \in C} u_j$ and

$$\omega_i(h_C)(u^\star, \overline{u}) = \begin{cases} \dfrac{\overline{u_i} - u_i^\star}{2}, & \text{if } \overline{u_i} \leq \bigwedge\limits_{j \in C_{-i}} u_j^\star \\ \left(\bigwedge\limits_{j \in C_{-i}} u_j^\star - u_i^\star \right) \left(1 - \frac{\bigwedge_{j \in C_{-i}} u_j^\star - u_i^\star}{2(\overline{u_i} - u_i^\star)} \right), & \text{if } u_i^\star < \bigwedge\limits_{j \in C_{-i}} u_j^\star < \overline{u_i} \\ 0, & \text{otherwise} \end{cases}$$

with $C_{-i} = C \setminus \{i\}$.

Proof: The Choquet integral, when expressed with respect to the Möbius transform, becomes (Grabisch, 1997):

$$C_\mu(u) = \sum_{C \subset \mathcal{N}} m(C) h_C(u)$$

Hence, by linearity of ω_i w.r.t. $\mathcal{H}$, we can deduce the following formula:

$$\omega_i(C_\mu)(u^\star, \overline{u}) = \sum_{C \subset \mathcal{N}} m(C) \times \omega_i(h_C(u^\star))(u^\star, \overline{u})$$

One clearly has for any $C \in \mathcal{N}$:

$$\omega_i(h_C)(u^\star, \overline{u}) = \frac{1}{\overline{u_i} - u_i^\star} \int_{u_i^\star}^{\overline{u_i}} \left[\left(s \wedge \bigwedge_{j \in C_{-i}} u_j^\star \right) - \bigwedge_{j \in C} u_j^\star \right] ds$$

Depending on the different cases, we obtain:

- If $u_i^\star \geq \bigwedge_{j \in C_{-i}} u_j^\star$:

$$\omega_i(h_c)(u^\star, \overline{u}) = \frac{1}{\overline{u_i} - u_i^\star} \int_{u_i^\star}^{\overline{u_i}} \left(\bigwedge_{j \in C_{-i}} u_j^\star - \bigwedge_{j \in C_{-i}} u_j^\star \right) ds = 0$$

- If $u_i^\star < \bigwedge_{j \in C_{-i}} u_j^\star$:
  If $\overline{u_i} \leq \bigwedge_{j \in C_{-i}} u_j^\star$

$$\begin{aligned} \omega_i(h_C)(u^\star, \overline{u}) &= \frac{1}{\overline{u_i} - u_i^\star} \int_{u_i^\star}^{\overline{u_i}} (s - u_i^\star) ds \\ &= \frac{\overline{u_i} - u_i^\star}{2} \end{aligned}$$

  If $\overline{u_i} > \bigwedge_{j \in C_{-i}} u_j^\star$

$$\begin{aligned} \omega_i(h_C)(u^\star, \overline{u}) &= \frac{1}{\overline{u_i} - u_i^\star} \left[\int_{u_i^\star}^{\bigwedge_{j \in C_{-i}} u_j^\star} (s - u_i^\star) ds + \int_{\bigwedge_{j \in C_{-i}} u_j^\star}^{\overline{u_i}} \left(\bigwedge_{j \in C_{-i}} u_j^\star - u_i^\star \right) ds \right] \\ &= \frac{1}{\overline{u_i} - u_i^\star} \left[-\frac{(u_i^\star - \bigwedge_{j \in C_{-i}} u_j^\star)^2}{2} + \bigwedge_{j \in C_{-i}} u_j^\star (\overline{u_i} - u_i^\star) - u_i^\star (\overline{u_i} - u_i^\star) \right] \\ &= \left(\bigwedge_{j \in C_{-i}} u_j^\star - u_i^\star \right) \left(1 - \frac{\bigwedge_{j \in C_{-i}} u_j^\star - u_i^\star}{2(\overline{u_i} - u_i^\star)} \right) \end{aligned}$$

□

Hence, the calculation of the integral of Eq. (4) gives the two equations of Proposition 2, that are more suitable for implementation.

The choice of the criterion with the highest value for the average improvement indicator applied to the Choquet integral is made by the averageImprovement_h function in the MCS framework.

6. Instantiating MCS in the search for an optimal solution

Depending on the definition of its components, the MCS framework can be used to find an optimal solution or to perform a partial search. In this section, we propose several instances of the algorithm that can be used to perform a complete search on a multicriteria combinatorial optimisation problem. We first introduce the property of *weak completeness* that allows us to classify the mono-criterion searches. In the following parts, we introduce two instances of the <setLocalConstraints> function and one stopping condition. Then, according to the *Space Reduction Properties* and the *Termination Property* introduced in Section 3.2.3, we

show that, when these functions are used together in the MCS framework, at the end of the algorithm it always returns an optimal solution if it exists.

6.1. Weak completeness of a mono-criterion search

As previously explained, many kinds of search strategies can be used in MCS for mono-criterion searches. A search on a criterion may produce an optimal solution, stop after the first solution found, follow a partial search scheme, etc. In optimisation, a search algorithm is said to be complete if, when it terminates, it has either found an optimal solution or proved that no solution exists. In practice, the MCS algorithm alternates the construction of several search trees. From a more general point of view, the completeness of the algorithm depends directly on the property of the search strategies that compose it.

In our case, we introduce the notion of *weak completeness* to distinguish clearly between the properties of the mono-criterion searches and the completeness of the whole algorithm:

Definition 5 (*Weak completeness).* A search algorithm is said to be weakly complete if it returns at least one solution if a solution exists.

In MCS, successive mono-criterion searches explore the search space and our aim is to guarantee the completeness of the overall algorithm. However, mono-criterion searches can be submitted to limits or local constraints.

In this context, the weak completeness property of mono-criterion search is essential. Indeed, when a weakly complete search that has not been subject to a local constraint cannot find any solution, then we can conclude that, either a solution had been found during the previous search and it is optimal, or there is no solution to the problem. Nevertheless, if a local constraint has been set before the search, the only possible conclusion is that there is no solution that simultaneously improves the overall evaluation and respects the local constraint.

A search strategy is said to be weakly complete when it defines a weakly complete search algorithm.

6.2. Local constraints

A local constraint creates a cut in the search space which remains during only one search. This kind of cut makes it possible to temporarily concentrate the search in some places where good solutions are believed to be. This can remove some solutions, but the temporary aspects of these constraints still make the definition of complete searches possible.

In the MCS framework, the function that does not set any local constraint is denoted noConstraint_h. For complete search, we propose to use only *improvement constraints* on the selected criterion.

Let $s^\star$ be the last solution found and c the criterion selected by the <getNextCriterion> function. Variable u_c models the satisfaction on criterion c that should be maximised during the next search. Function getValue($s^\star$, c) returns the value of u_c in solution $s^\star$.

It is important to point out that the selection of a criterion by the criterion search heuristic indicates a strategy that is efficient in finding good solutions on a criterion. However, it does not impose the search to improve the selected criterion until it finds a first solution. Thus, especially when the construction of new solutions becomes harder, it can be useful to explicitly set the constraint $u_c >$ getValue($s^\star$, c) by means of a local constraint. We call this kind of constraint an *improvement constraint* on the selected criterion.

We propose two functions based on this idea. They differ from the frequency at which the $u_c > \mathsf{getValue}(s^\star, c)$ constraint is posted to the model:

- In the systematicImp_h function, the constraint is posted before each search.
- In the consecutiveImp_h function, the constraint is posted only when the criterion c has also been selected to guide the previous search. This local constraint can be particularly convenient when the mono-criterion searches are limited to one solution.

According to the requirements of the framework, these two functions return true when they set a local constraint and false otherwise.

6.3. A stopping condition for complete search

When MCS is used with *weakly complete strategies* and *improvement constraints on the selected criterion*, the <checkTermination> component can be defined such that MCS always returns an optimal solution (if a solution exists).

This corresponds to the optimalStoppingCondition function described in Fig. 4.

We recall that the stopping condition returns the true value to interrupt the search. The *local* variable indicates if a local constraint was set before the previous search, $s^\star$ is the best solution found so far and s is the result of the last search (note that $s \neq s^\star$ only if $s = \mathsf{nil}$). As before, y models the overall evaluation, vector u contains the utility variables and c is the index of the criterion that guided the last search.

We introduce the $\mathsf{status}(s)$ function, which returns the status of the solution s with respect to the last search:

- $\mathsf{status}(s) = \mathsf{false}$: maximize proved that there was no solution to the problem solved during the last search.

```
optimalStoppingCondition(local, s*, s, y, u, c)
    if (c = 0)
        return false
    else if (status(s) = true)
            setConstraint(u_c ≤getValue(s, c))
        endif
        if (getOverallValue(s*) ≥ ȳ)
            return true
        else if (s = nil)
                if (local)
                    setConstraint(u_c ≤getValue(s*, c))
                    if (getOverallValue(s*) ≥ ȳ)
                        return true
                    else return false
                else return true
            else return false
        endif
    endif
end
```

Fig. 4 MCS stopping condition for complete search

- status(s) = true: maximize found at least one solution and proved that there is no solution that is better on both the criterion u_c and the overall evaluation y (i.e. if we impose ($y \geq$getOverallValue(s)), s is optimal for criterion c).
- status(s) = unknown: either maximize found a solution but did not prove its optimality, or no solution has been found, (s = nil), but it has not been proved that there was no solution to the problem (i.e., the search tree has not been completely explored).

First of all, optimalStoppingCondition systematically returns false when $c = 0$, which corresponds to the first iteration of the algorithm. It stops the algorithm when the optimality of the last found solution is verified (getOverallValue($s^\star$) $\geq \overline{y}$). When status(s) equals true, it is possible to inform the model that there is no solution that is better on both the last selected criterion c and the overall evaluation (Section 3.2.3: RP1): this allows us to decrease the upper bound of u_c and possibly, to reduce the search space. To achieve this, we use getValue(s, c), which returns the satisfaction degree of s on criterion c (i.e., the u_c value for s), and the setConstraint function to reduce the upper bound of u_c. Otherwise, if no solution has been found (s = nil), two cases can arise: either a local constraint was set before the previous search and we can decrease $\overline{u_c}$ (since the constraint was an improvement constraint on c), or the search was not submitted to any constraint and the algorithm stops. When a search returns a solution, the algorithm continues.

Proposition 3 (Optimality of the last found solution). *Suppose an instance of the MCS algorithm such that: the strategies defined for each criterion are complete search strategies, the criterion choice heuristic is based on an indicator that satisfies (M) and (NG), the local constraint function is either* systematicImp_h, consecutiveImp_h *or* noConstraint_h *and the stopping condition is* optimalStoppingCondition. *Then this algorithm always terminates and the last solution found is optimal with respect to the multicriteria aggregation function of the problem.*

Proof: In this proof, we first show that, if it exists, MCS necessarily returns an optimal solution when used with the specified parameters. In the second part, we show that the algorithm cannot loop on the same unsuccessful search and that it terminates in finite time.

First, by construction:

(i) Condition (getOverallValue($s^\star$) $\geq \overline{y}$) checks if the overall evaluation $y^\star$ of the last solution found $s^\star$ corresponds to the maximum possible satisfaction level $\overline{y}$. In this case MCS terminates and returns $s^\star$(Section 3.2.3: TP1). Otherwise the algorithm continues while the searches find some solutions ($s \neq$ **nil**).

(ii) If no solution is found by a mono-criterion search ($s =$ **nil**), there are two alternatives:

  (a) $local$ = true: a local constraint was used. Since the local constraint is an improvement constraint on the selected criterion c and since the strategies are weakly complete, the upper bound of u_c can be decreased to the last value obtained on this criterion ($\overline{u_c} = u_c^\star$) (Section 3.2.3: RP2), but the search continues.
  (b) $local$ = false: the weak completeness of the strategies allows us to say that no solution satisfies the $y > y^\star$ constraint and therefore the last solution found is optimal (Section 3.2.3: TP1).

Second, we show that MCS cannot loop on a criterion that cannot be improved. Indeed, when optimalStoppingCondition returns false, either the last search found a solution and $y^\star$ has been improved, or the last search was subject to a local constraint and did not find any solution, in which case, $\overline{u_c} = u_c^\star$. Therefore, when MCS enters the criterion choice heuristic,

according to (NG), $w_c(\mathcal{H})(u^\star, \overline{u}) = 0$. Let us show that this criterion cannot be selected for the next search. Let $I = \{i \in \mathcal{N} |\ w_i(\mathcal{H})(u^\star, \overline{u}) > 0\}$, the set of criteria that would be selected in priority with respect to c. By contradiction, we show that $I \neq \emptyset$. Assume indeed that $I = \emptyset$. Therefore, by (G), $\forall i \in \mathcal{N}, \overline{u_i} = u_i^\star$. Hence $\overline{y} = y^\star$, which makes the algorithm stop and contradicts the hypothesis that MCS entered the criterion choice heuristic. Hence, $I \neq \emptyset$ and there exists a criterion $j \neq c$ such that $\overline{u_j} > u_j^\star$. □

6.4. Experimentation

We experimented the MCS algorithm on small instances of the examination timetabling problem. Several instances of MCS were tested in order to evaluate the advantages and drawbacks of the MCS framework components.

6.4.1. The examination timetabling problem

Given a set of examinations, a set of students each enrolled for a given list of examinations, a set of rooms of fixed capacities and a set of periods, the examination timetabling problem consists in assigning a period and a room to each examination such that (i) two examinations that are given to a same student cannot be planned on the same period and (ii) the capacity of a room cannot be exceeded. We assume that as long as constraints (i) and (ii) hold, several examinations can take place in the same room at the same time but that the number of students attending an examination cannot be distributed over several rooms.

A simple multicriteria model has been constructed based on three attributes: the date of the last examination planned (criterion *duration of the examination*), the number of rooms used (criterion *rooms employment*) and the number of times a student has two consecutive examinations (criterion *spreading of the exams*). These criteria are aggregated using the Choquet integral (the whole multicriteria model is more precisely described in Le Huédé (2003)).

6.4.2. Instances

Small scenarios have been constructed in order to evaluate the performance of MCS in complete search. The main characteristics of these scenarios are described in the following table:

	Number of periods	Number of exams	Number of rooms	Number of students
Sc. 12	9	12	2	49
Sc. 15	9	15	3	56
Sc. 20	11	20	2	104

The algorithms are launched for various fuzzy measures that correspond to typical cases of aggregation functions. The $\mu_{\min}$ measure models an intolerant expert (i.e., complementary criteria), $\mu_{\max}$ models a tolerant expert and μ_{mean} models the case where criteria are independents. Although $\mu_{\min}$, $\mu_{\max}$ and μ_{mean} are respectively close to the min, max and the mean functions, they do not exactly model them.

The coefficients of a fuzzy measure μ are given in the following order: $\mu = \{\mu(\emptyset), \mu(\{1\}), \mu(\{2\}), \mu(\{1, 2\}), \mu(\{3\}), \mu(\{1, 3\}), \mu(\{2, 3\}), \mu(\{1, 2, 3\})\}$.

- $\mu_{\min} = \{0., 0.2, 0.2, 0.21, 0.2, 0.21, 0.21, 1.\}$

- $\mu_{\max} = \{0., 0.8, 0.8, 0.9, 0.8, 0.9, 0.9, 1.\}$
- $\mu_{\text{mean}} = \{0., 0.2, 0.2, 0.8, 0.2, 0.8, 0.8, 1.\}$

6.4.3. Strategies

To find a solution, the search tree assigns a date and a room number to each examination. Three strategies, st_1, st_2, st_3 were designed for criteria duration of the examination, room employment and spreading of the exams respectively;

- *Variable choice heuristics:* In all strategies the examinations are sorted in decreasing order with respect to the number of examinations they are in conflict with (two examinations are said to be in conflict when they cannot be planned at the same time, i.e. when a student registered for both of them). When two exams are in conflict with a same number of exams, they are sorted according to the number of students taking each exam (larger exams are instantiated first). Then, for the duration (st_1) and the spreading (st_3) criteria, the date variable is instantiated before the room variable and conversely, for the rooms employment (st_2) criterion, the room is instantiated first.
- *Labelling strategies:* the date variables are instantiated in increasing order for the duration (st_1) and the rooms employment (st_2) criteria. For the spreading of the exams (st_3), we use a look ahead heuristic of depth one that selects the most satisfying date according to this criterion (i.e.: the current node is completely developed on a depth of one and the most promising branch w.r.t. the spreading criterion is selected). For the room variables, the labelling strategy is the same for all the strategies: it chooses the largest room in priority.

6.4.4. Results

We compare eight instances of the MCS algorithm with the performances of st_1, st_2 and st_3 when used separately to optimise the overall evaluation in one single tree search, as in classical mono-objective optimisation. We also tested a mono-objective approach that uses a look-ahead labelling strategy, using the upper bound of the overall evaluation to evaluate each period in the date variables instantiation (this strategy is denoted LA).

The different MCS instances are presented in Fig. 5. In the *opt* strategy, each mono-criterion search finds an optimum for its criterion. *nbSolLimit* indicates that the searches stop when they have found a given number of solutions (i.e.: one solution for the duration and room employment criteria and two solutions for the spreading of the exams). These algorithms use the optimalStoppingCondition function and return an optimal solution.

The performances of these algorithms were evaluated on four instances of the timetabling problem (Fig. 6). For each instance we compare the total completion time. The average of

Fig. 5 Instances of the MCS framework for complete search

	Strategies	Indicator	local constraint
mcs_1^{ω}	opt	$\omega_i(\mathcal{H})(u^\star, \overline{u})$	noConstraint_h
mcs_2^{ω}	$nbSolLimit$	$\omega_i(\mathcal{H})(u^\star, \overline{u})$	noConstraint_h
mcs_3^{ω}	$nbSolLimit$	$\omega_i(\mathcal{H})(u^\star, \overline{u})$	consecutiveImp_h
mcs_4^{ω}	$nbSolLimit$	$\omega_i(\mathcal{H})(u^\star, \overline{u})$	systematicImp_h
mcs_1^{χ}	opt	$\chi_i(\mathcal{H})(u^\star, \overline{u})$	noConstraint_h
mcs_2^{χ}	$nbSolLimit$	$\chi_i(\mathcal{H})(u^\star, \overline{u})$	noConstraint_h
mcs_3^{χ}	$nbSolLimit$	$\chi_i(\mathcal{H})(u^\star, \overline{u})$	consecutiveImp_h
mcs_4^{χ}	$nbSolLimit$	$\chi_i(\mathcal{H})(u^\star, \overline{u})$	systematicImp_h

Nb ex	st_1 (ms)	st_2 (ms)	st_3 (ms)	LA (ms)	mcs_1^ω (ms)	mcs_2^ω (ms)	mcs_3^ω (ms)	mcs_4^ω (ms)	mcs_1^χ (ms)	mcs_2^χ (ms)	mcs_3^χ (ms)	mcs_4^χ (ms)
	μ_{min}											
12	50	50	**10**	30	50	30	40	30	50	30	20	40
15	790	780	790	**570**	960	710	660	660	960	710	650	660
20	**16 020**	29 430	24 490	23 800	33 980	24 230	22 580	28 270	34 550	24 210	22 870	28 360
	μ_{mean}											
12	50	50	**30**	70	100	80	80	70	90	80	80	70
15	15 070	19 150	23 630	24 950	19 810	14 690	14 800	**14 520**	19 600	14 530	14 720	14 730
20	58 400	99 840	83 750	87 750	58 720	51 490	51 020	45 090	98 400	52 740	52 540	**44 800**
	μ_{max}											
12	30	30	**10**	40	20	20	20	20	20	20	20	20
15	2 390	4 220	1 840	3 460	56 850	1 170	**1 160**	1 190	57 080	**1 160**	**1 160**	1 190
20	8 210	8 290	**1 010**	12 930	11 140	2 620	2 650	17 900	11 180	2 630	2 660	17 560
	Synthesis											
avg	11 223	17 982	15 062	17 066	20 181	10 560	**10 334**	11 972	24 658	10 678	10 524	11 936
dist	2 542	9 301	6 381	8 386	11 500	1 879	**1 653**	3 291	15 978	1 998	1 843	3 256

Fig. 6 Time results for complete search with MCS

these performances and the average distance to the best performance are given in the avg and dist lines of the table respectively.

From these results we can say that, considering only the MCS algorithms, on these problem instances there are few differences between the ω and the χ indicators, and only a slight difference between noConstraint_h and consecutiveImp_h (although consecutiveImp_h seems to be generally a bit better than noConstraint_h). Considering the algorithms that use the *opt* strategies, searching for a solution that is optimal for the selected criterion at each search is worthless for every instance. Similarly, setting an improvement local constraint systematically before each search causes some large performance breakdowns in the $\mu_{\max}$ case. However, it gives good results when the aggregation functions come close to the weighted sum.

For the classical strategies, st_1 is better for $\mu_{\min}$ and μ_{mean} and st_3 outperform the others on $\mu_{\max}$.

If we compare all the algorithms we can see that the mcs_3 instances give the best results on average. In addition, these instances of the algorithm seem to be much more robust to changes in the aggregation function than single strategies. When they do not give the best solution, they are always quite close to it.

7. Partial search with MCS

When addressing real combinatorial optimisation problems, the search space is often so large that it is impossible to use complete optimisation algorithms. In this case, it is necessary to use a partial search algorithm, whose objective is to find the best possible solution within a given time contract. According to this objective, the classical Branch & Bound approach often offers poor performances: following the depth-first exploration strategy, the search easily gets stuck in the bottom of the tree. Due to the size of the problem, if the search strategy makes a bad choice on the top of the tree, it has little chance to backtrack on this decision before the end of the time contract.

As a result, several partial exploration strategies have been developed in order to favour backtracks on early decisions (Ginsberg, 1993; Beldiceanu et al., 1997; Harvey and Ginsberg, 1995; Gomes, Selman, and Kautz, 1998; Shaw, 1998). Among them, *iterative weakening* methods use limits on the development of the search tree to build new exploration strategies (de Givry and Jeannin, 2003). In this section, we show how such limits can be used in the strategies of the MCS framework.

7.1. Iterative weakening in MCS

7.1.1. Principles of iterative weakening methods

Iterative weakening methods are used in CP to perform partial searches on the search tree. They rely on limits that constrain the size of the explored part of a given search tree. The problem is solved iteratively with a progressively relaxed limit until the complete search tree has been explored.

The ToOLS library (de Givry and Jeannin, 2003) provides a large set of primitives to describe search trees and to apply limits on any sub-part of a tree. A limit determines if a node can be visited or not. It can be written as a condition *expression* $\leq$ *threshold*, where *expression* defines a way to evaluate a node (primitive nodelimit), a path (primitive pathlimit) or a subtree (treelimit and globallimit) during the search. Thus, these primitives allow us to constrain:

- *the rank of a visited node* (nodelimit): at a node of the search tree, the child nodes are sorted by the labelling strategy and assigned a rank. The first choice starts at rank zero.
- *the distance from a node to the preferred node* (nodelimit).
- *the number of leaves, the number of backtracks or the number of nodes in a subtree* (treelimit and globallimit).
- *the sum of all node ranks or the sum of all the node distances in a search path* (pathlimit).

As detailed in de Givry and Jeannin (2003), a large set of partial search algorithms can be described by means of limit primitives applied to search strategies.

7.1.2. Application to MCS

As introduced in Section 4.2, a mono-criterion search in MCS can follow any exploration strategy. However, when using MCS in the context of a partial search, under a time limit, two aspects have to be taken into account: First of all, in order to keep a good robustness property, it is necessary to allow each strategy equal opportunities to be used. Second, when it gets hard to find a solution that improves the overall evaluation, it may be useful to favour a strategy that still can find some solutions.

To fulfill these requirements, we propose using the following principles for creating a new instance of MCS.

1. A search strategy on a criterion c is subject to a limit l_c, initialised to 0 on the first iteration.
2. When a mono-criterion search on c cannot find any solution, its limit l_c is increased for the next search.
3. Between two iterations of MCS, the criterion choice heuristic gives priority to the criterion with the smallest value for l_c.

Thus, mono-criterion searches are subject to very restrictive limits on the beginning of the algorithm. If a search on c cannot find any solution, the limit is relaxed, but since the criterion choice heuristic takes l into account, the other strategies have to be tested before a strategy for criterion c can be used again. In addition, since the limit is not increased when a search returns a solution, we favour the use of a strategy that find solutions when the improvement of the overall evaluation becomes difficult.

These generic principles can be applied to any partial search strategy that uses a single limit. They are described in the **incLimitStoppingCondition** stopping condition and in the **smallestPathimit_h** heuristic, presented in the following paragraphs.

7.2. Stopping condition

The dynamic modification of search strategies is performed in the stopping condition of the algorithm. The algorithm of **incLimitStoppingCondition** is described in Fig. 7. It supposes that the user supplies strategies without limits.

In this algorithm, the vector l contains the limits associated with each strategy.

On the first iteration of the MCS algorithm ($c = 0$), the limits are initialised to 0 by the **initLimits** function and the initial strategies are saved by **saveInitialStrategies**. Thus, the strategy that describes the complete search tree for a criterion i will be available by calling the **getInitialStrategy**(i) function. Furthermore, to associate a strategy ST to a criterion i, we use **setStrategy**(i, ST) and **setLimit**(d, ST) to apply a limit equal to d to the ST strategy.

After each iteration of MCS, the stopping condition is very similar to **optimalStoppingCondition**, except when an incomplete search on criterion c does not return any solution

```
incLimitStoppingCondition(local, s*, s, y, u, c, l)
    if (c = 0)
        pl ← initLimits(n)
        saveInitialStrategies()
        for i = 1 to n
            setStrategy(i, setLimit(l_i, getInitialStrategy(i)))
        endfor
        return false
    else if (getOverallValue(s*) ≥ ȳ)
            return true
        else if (s = nil)
                if (status(s) = false)
                    if (local)
                        setConstraint(u_c ≤getValue(s*, c))
                        if (getOverallValue(s*) ≥ ȳ)
                            return true
                        else return false
                    else return true
                else
                    l_c ← l_c + 1
                    setStrategy(c, setLimit(l_c, getInitialStrategy(c)))
                endif
                return false
            endif
        endif
    endif
end
```

Fig. 7 A stopping condition for partial search

```
smallestLimit_h(l, n)
    smallestL ← l_1, smallestIdx ← 1
    for i = 2 to n
        if (l_i < smallestL)
            smallestL ← l_i
            smallestIdx ← i
        endif
    endfor
    return smallestIdx
end
```

Fig. 8 The smallestLimit_h criterion choice heuristic

$((s = \textbf{nil})$ and $(\textbf{status}(s) = \textbf{unknown}))$. In this case, the limit l_c is increased and its corresponding strategy is updated.

7.3. Criterion choice heuristic

The criterion choice heuristic that chooses the criterion with the smallest value for l is called **smallestLimit_h** (Fig. 8).

A very straightforward improvement of **smallestLimit_h** can be obtained by using a worth to improve indicator to distinguish between equivalent criteria. Thus, we also propose the **smallestLimitMaxImp_h** that merge the **smallestLimit_h** algorithm with **maxImprovement_h**. When several criteria have the smallest limit, the heuristic uses the criterion with the lowest value for the χ_i indicator (Section 5.2).

```
smallestLimitMaxImp_h(s*, l, n)
    smallestL ← l_1
    smallestIdx ← 1
    worth ← χ_1(H)(u*, ū)
    for i = 2 to n
        if (l_i ≤ smallestL)
            if ((l_i = smallestL) and (χ_i(H)(u*, ū) > worth))
                smallestL ← l_i
                smallestIdx ← i
                worth ← χ_i(H)(u*, ū)
            endif
        else
            smallestL ← l_i
            smallestIdx ← i
            worth ← χ_i(H)(u*, ū)
        endif
    endfor
    return smallestIdx
end
```

Fig. 9 The smallestLimitMaxImp_h criterion choice heuristic

7.4. Experimentation

We experimented the MCS algorithm for partial search on large instances of the timetabling examination problems. These instances were built from the data proposed by Carter et al. by adding 3 rooms of limited capacities to the initial problems (first introduced in Lee, Carter, and Laporte (1996), the problem solved by Carter et al. was a satisfaction problem). We also relaxed the initial constraint on the timetabling duration and studied the three criteria optimisation problem introduced in Section 6.4.1.

7.4.1. Instances and strategies

The main characteristics of these instances are given in the following table:

Scenario	University	Nb. of periods	Nb. of exams	Nb. of students
HEC	Ec. des Htes Études Commerciales	22	80	2823
STA	St. Andrews High school	40	138	549
YOR	York Mills Collegiate	30	180	919
UTE	Univ. of Toronto, Engineering	30	184	2750
EAR	Earl Haig Collegiate	30	189	1108
TRE	Trent University	35	261	4360

The MCS algorithm uses the **consecutiveImp_h**, **smallestLimitMaxImp_h** and **incLimitStoppingCondition** with a pathlimit on the sum of all ranks in a search path. For each instance and each fuzzy measure introduced for complete search, the algorithm is given 10 minutes to find the best possible solution. For comparison, the same time is given to four algorithms that use the st1, st2, st3 and LA strategies defined in Section 6.4.3. These search strategies are combined to an LDS exploration strategy (Harvey and Ginsberg, 1995), which is one of the most efficient exploration strategies for partial search.

7.4.2. Results

Figure 10 presents the value of the overall evaluation returned by each algorithm.

Considering first the static heuristics in these experiments, we observe that strategy st1, which had very good results in complete search, performs very badly in its LDS version on large problems, whereas LDS(st2) finds the best solution on some instances, but is often very far from the best results on the other instances.

Regarding especially the results at 10 mn, look ahead based heuristics seem to perform much better: strategy LDS(st3), which uses a look ahead heuristic for the spreading criterion, obtains very good results, as well as strategy LDS(LA) which is "guided" by the overall evaluation. These two strategies need however much more time for finding a first solution. On 2 out of the 5 instances, LDS(st3) systematically fails to find any solution within 1 minute and LDS(LA) is even worse.

The analysis of MCS results shows that it does not outperform the best of the 4 reference algorithms on any problem. Nevertheless, it always succeeds in finding a solution before 1 minute and the quality of the solution returned by MCS is often very close to the quality of the best solution found.

Iterative weakening algorithms such as LDS already offer a degree of diversification that makes them reach quite different parts of the search space. Nevertheless, on large problems such as those considered in our experiments, it is still hard for any of the single strategy

	time	LDS(LA)	LDS(st1)	LDS(st2)	LDS(st3)	MCS
Scenario	μ_{min}					
HEC	1 mn	0.146370	0.084239	0.145914	**0.147483**	0.146833
	10 mn	0.155663	0.095253	**0.156395**	0.155137	0.155137
STA	1 mn	-	0.029222	0.029437	**0.163563**	0.163310
	10 mn	0.137067	0.029232	0.029461	**0.172096**	0.171239
YOR	1 mn	-	0.088432	**0.137455**	-	0.137425
	10 mn	0.097980	0.098059	0.137874	**0.162376**	0.161012
UTE	1 mn	0.909462	-	0.239623	**0.910170**	0.726495
	10 mn	**0.982932**	-	0.240211	0.946885	0.799965
EAR	1 mn	-	0.204523	**0.232026**	-	0.231965
	10 mn	0.555518	0.204630	0.232132	**0.633314**	0.596165
	μ_{mean}					
HEC	1 mn	0.205359	0.088542	0.148879	**0.356719**	0.325795
	10 mn	0.271977	0.098798	0.215366	**0.399411**	0.335257
STA	1 mn	-	0.033882	0.034634	**0.178763**	**0.178763**
	10 mn	0.110644	0.034635	0.038612	**0.186692**	**0.186692**
YOR	1 mn	**0.295636**	0.202310	0.250362	-	0.250355
	10 mn	**0.304821**	0.242092	0.250385	0.226955	0.250358
UTE	1 mn	**0.946632**	-	0.771376	0.943800	0.906600
	10 mn	**0.974643**	-	0.793063	0.953080	0.915900
EAR	1 mn	-	0.590099	**0.658210**	-	**0.658210**
	10 mn	0.645350	0.590124	0.658262	**0.675794**	0.666080
	μ_{max}					
HEC	1 mn	0.419116	0.209271	0.449445	**0.465514**	**0.465514**
	10 mn	**0.508472**	0.249664	0.498875	0.500052	0.498745
STA	1 mn	0.564525	0.042362	0.043118	**0.610811**	0.604811
	10 mn	0.605667	0.042530	0.043850	**0.636525**	**0.636525**
YOR	1 mn	-	0.231848	**0.422909**	-	0.422905
	10 mn	0.315978	0.273315	**0.422921**	0.316569	0.422905
UTE	1 mn	0.970641	-	0.884646	**0.983100**	0.964500
	10 mn	**0.994186**	-	0.888866	0.987670	0.978450
EAR	1 mn	-	0.714851	**0.828569**	-	0.828555
	10 mn	0.699730	0.714862	**0.828582**	0.712334	0.828557

Fig. 10 Overall evaluation of the best solution found after 1 and 10 mn

algorithm to find very distant solutions. This makes the quality of the returned solution deeply connected to the quality of the search strategy. As previously stated in the introduction of this paper, when the objective function of the problem is based on several contradictory criteria, defining a search strategy that is good in all cases becomes nearly impossible. In our experiments, a further look over the runs of the algorithms shows that strategies LDS(st1) and LDS(st2) cannot find any solution of good quality for the third criterion, whereas LDS(st3) always optimises the spreading of the examination and gives bad results on other criteria. Of course, this comes from the fact that minimising the duration of the examinations and the number of used rooms is closely contradictory with the minimisation of the number of times a student has two consecutive examinations. The size of the problem makes finding a good trade-off a lot harder. Thus, even the more holistic approach followed by LDS(LA) often fails to give better results after a long time.

Finally, within a limited time MCS manages to take some good properties in each of its mono-criterion strategies. It performs a more diversified exploration and is less sensitive to contradictory effects between the criteria and to differences between instances or preference relations. In our experiments, both the speed of static heuristics and the capacities of dynamic heuristics for finding good solutions for complex criteria were exploited by MCS.

In conclusion, we believe that these first experiments of MCS for local search are promising and that further work, on local constraints for example, will enable us to make a better combination of the different strategies and to reach even better solutions.

8. Conclusion

In this paper we introduced the MCS framework for defining search strategies in the context of a multicriteria optimisation problem. Indeed the definition of a search strategy in a classical Branch & Bound search is often made difficult in multicriteria optimisation due to the influence of several conflicting criteria on the overall evaluation. The main idea of this framework is that it is easier to define a good search strategy per criterion than a good overall strategy. According to this hypothesis, the MCS framework offers the suitable components to facilitate the definition of such algorithms for multicriteria optimisation problems. In our experiments on the examination timetabling problems, we showed that an adequate use of these strategies allows the user of MCS to build more efficient complete search algorithms.

In the final part, we introduced our latest development on MCS applied to partial search. We proposed new components that perform a progressive relaxation of limits on the mono-criterion strategies. We showed that, on real-size instances of the timetabling examination problem, MCS was more robust than state-of-the-art partial search algorithms. These preliminary results suggest that the MCS framework gives a better control for the selection of the explored parts of the search space and that more efficient search algorithms can be designed by specialising this general scheme.

Acknowledgments We would like to thank our anonymous reviewers for their valuable and detailed comments.

References

PLATON Team (2001). "Eclair Reference Manual, Version 6." Technical Report Platon-01.16, THALES Research and Technology, Orsay, France.

Bana e Costa, C.A. and J.C. Vansnick. (1994). "A Theoretical Framework for Measuring Attractiveness by a Categorical Based Evaluation TecHnique (MACBETH)." In *Proc. XIth Int. Conf. on MCDM*, pp. 15–24. Coimbra, Portugal.

Barichard, V. and J.-K. Hao. (2003). "A Population and Interval Constraint Propagation Algorithm." In *LNCS 2632*, pp. 88–101. Springer.

Beldiceanu, N., E. Bourreau, H. Simonis, and D. Rivreau. (1997). "Partial Search Strategy in CHIP." In *Proc. 2nd Int. Conf. on Meta-Heuristics*. Sophia-Antipolis, France.

Chankong, V. and Y.Y. Haimes. (1983). *Multiobjective Decision Making: Theory and Methodology*. North-Holland.

Choquet, G. (1953). "Theory of capacities." *Annales de l'Institut Fourier*, 5, 131–295.

de Givry, S. and L. Jeannin. (2003). "TOoLS: A Library for Partial and Hybrid Search Methods." In *Proceedings of CP-AI-OR'03*, Montral. Canada.

Ehrgott, M. and X. Gandibleux (Eds.) (2002). *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. Boston: Kluwer Academic Publishers. ISBN 1-4020-7128-0.

Focacci, F. and D. Godard. (2002) "A Practical Approach to Multi-Criteria Optimization Problems in Constraint Programming." In N. Jussien and F. Laburthe (Eds.), In *Proceedings of CP-AI-OR'02*, pp. 65–75, Le Croisic, France.

Gavanelli, M. (2002). "An Algorithm for Multi-Criteria Optimization in CSPs." In Frank van Harmelen, (Ed), In *Proceedings of ECAI-2002*. pp. 136–140. Lyon, France, IOS Press.

Ginsberg, M.L. (1993). "Dynamic Backtracking." *Journal of Artificial Intelligence Research*, 1, 25–46

Ginsberg, M.L. and W.D. Harvey. (1992). "Iterative Broadening." *Artificial Intelligence*, 55, 367–383.

Gomes, C.P., B. Selman, and H. Kautz. (1998) "Boosting Combinatorial Search Through Randomization." In *Proc. of AAAI-98*, Madison, WI.

Grabisch. M. (1995). "Fuzzy Integral in Multicriteria Decision Making." *Fuzzy Sets & Systems*, 69, 279–298.

Grabisch, M. (1996). "The Application of Fuzzy Integrals in Multicriteria Decision Making." *European J. of Operational Research*, 89, 445–456.

Grabisch, M. (1997). "Alternative Representations of Discrete Fuzzy Measures for Decision Making." *Int. J. of Uncertainty, Fuzziness, and Knowledge Based Systems*, 5, 587–607.

Grabisch, M. and C. Labreuche. (2001). "How to Improve Acts: An Alternative Representation of the Importance of Criteria in MCDM." *Int. J. of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 9(2), 145–157.

Grabisch, M., T. Murofushi, and M. Sugeno. (2000). *Fuzzy Measures and Integrals. Theory and Applications (edited volume)*. Studies in Fuzziness. Physica Verlag.

Grabisch, M. and M. Roubens. (2000). "Application of the Choquet Integral in Multicriteria Decision Making." In M. Grabisch, T. Murofushi, and M. Sugeno (Eds.), *Fuzzy Measures and Integrals—Theory and Applications*, pp. 348–374. Physica Verlag.

Harvey, W.D. and M.L. Ginsberg. (1995). "Limited Discrepancy Search." In C.S. Mellish (Ed.), *Proc. of IJCAI-95*, pp. 607–613. Montréal Canada, Morgan Kaufmann.

Junker, U. (2002). "Preference-Based Search and Multi-Criteria Optimization." In *Proceedings of AAAI-02*, pp. 34–40. Edmonton, Alberta, Canada.

Keeney, R.L. and H. Raiffa. (1976). *Decision with Multiple Objectives*. New York: Wiley.

C. Labreuche. (2004). "Determination of the Criteria to be Improved First in Order to Improve as Much as Possible the Overall Evaluation." In *Proceeding of IPMU 2004*, pp. 609–616. Perugia, Italy.

Labreuche, C. and M. Grabisch. (2003). "The Choquet Integral for the Aggregation of Interval Scales in Multicriteria Decision Making." *Fuzzy Sets and Systems*, 137(1), 11–26.

Le Huédé, F. (2003). "Intégration d'un modèle d'Aide à la Décision Multicritère en Programmation Par Contraintes." PhD thesis, Universitée Paris 6.

Le Huédé, F., P. Gérard, M. Grabisch, C. Labreuche, and P. Savéant. (2002). "Integration of a Multicriteria Decision Model in Constraint Programming." In B. Brabble, J. Koehler, and I. Refanidis, (Eds.), In *Proceedings of the AIPS'02 Workshop on Planning and Scheduling with Multiple Criteria*, pp. 15–20. Toulouse, France.

Lhomme, O. (1993). "Consistency Techniques for Numerical CSPs." In *Proceedings of IJCAI 1993*, pp. 232–238. Chambery, France.

Marichal, J.L. (1998). "Aggregation Operators for Multicriteria Decision Aid." PhD thesis, University of Liège.

Lee, S.Y., M.W. Carter, and G. Laporte. (1996). "Examination Timetabling: Algorithmic Strategies and Applications." *Journal of the Operational Research Society*, 47, 373–383.

Shaw, P. (1998). "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems." In *Proc. of CP-98*, pp. 417–431, Pisa, Italy.

Sugeno, M. (1974). "Theory of Fuzzy Integrals and its Applications." PhD thesis, Tokyo Institute of Technology

Walsh, T. (1997). Depth-Bounded Discrepancy Search. In *Proc. of IJCAI-97*, pp. 1388–1395, Nagoya, Japan.