Operations Research Perspectives 8 (2021) 100177

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

Multi-objective evolutionary search strategies in constraint programming

Robert Bennetto *, Jan H van Vuuren

Stellenbosch Unit for Operations Research, Department of Industrial Engineering, Stellenbosch University, Stellenbosch, South Africa

ARTICLE INFO

Keywords:
Combinatorial optimization
Multi-objective optimization
Genetic algorithms
Constraint programming

ABSTRACT

It has been shown that evolutionary algorithms are able to construct suitable search strategies for classes of *Constraint Satisfaction Problems* (CSPs) in Constraint Programming. This paper is an explanation of the use of multi-objective optimisation in contrast to simple additive weighting techniques with a view to develop search strategies to classes of CSPs. A hierarchical scheme is employed to select a candidate strategy from the Pareto frontier for final evaluation. The results demonstrate that multi-objective optimisation significantly outperforms the single objective scheme in the same number of objective evaluations. In situations where strategies developed for a class of problems fail to extend to unseen problem instances of the same class, it is found that the structure of the underlying CSPs do not resemble those employed in the training process.

1. Introduction

Constraint Programming (CP) is a declarative paradigm for defining discrete optimisation or satisfiability problems. A problem instance (satisfiability or optimisation) is referred to as a *Constraint Satisfaction Problem* (CSP). Larger, or more complex CSPs often require state-of-the-art heuristic strategies to improve search performance. While a heuristic strategy may be employed, the overall CP search remains a *complete search*, in that, given sufficient time, the search will terminate with either an optimality, feasibility or infeasibility proof.

It has been shown by several authors such as Minton [13], Epstein et al. [7] and Bain et al. [2] that heuristic approaches or metaheuristics can be used to develop search algorithms for solving classes of CSPs effectively. These approaches employ a single objective in the measurement of the quality of solutions found, even though multiple measurements as to the quality of an incomplete search may, in fact, be taken. As an example, the metrics presented by Schuurmans and Southey [16] are not necessarily directly comparable to one another as the units of measurement for the metrics vary (mobility, coverage, depth, flips). If such metrics were used in a single objective scheme, a weighting for each of the metrics would be required in order to define an explicit trade-off between objective function components for use in single objective metaheuristics.

While the approach of finding good branching strategies is heuristic, the underlying CP solver remains exact, which allows for the calibration of the solver for a class of problems that may result in significant performance improvements carried forward to unseen problem instances. In real-world applications, the methodology presented in this paper allows operations researchers to create heuristics well-suited to classes of problems in order to reduce future computational burden.

The aim of this paper is to address a weakness of the *simple additive weighting* (SAW) metaheuristic scheme employed by Bennetto and Van Vuuren [3] and proposes a multi-objective formulation of the metaheuristic search for a suitable CP branching strategy to solve a class of CSPs. The multi-objective approach demonstrates a statistically significant improvement over a SAW scheme across the same objective dimensions for an equivalent number of objective function evaluations. The concepts of CSPs, *Genetic Programming* (GP) and multi-objective optimisation are formally introduced in Section 2, while the methodology employed by the GP to find high-quality branching strategies is described in Section 3. Section 4 details the empirical results of the GP and includes an explanation of the success achieved by the multi-objective methodology. Conclusions are finally provided in Section 5.

2. Background

2.1. Constraint programming

A constraint within CP may be defined formally as follows [4].

Definition 1. (*Constraint*) A constraint c is a relation defined on a sequence of variables $X(c) = (x_{i_1}, \ldots, x_{i_{|X(c)|}})$, called the scheme of c. Here c is the subset of $\mathbb{Z}^{|X(c)|}$ which contains the combinations of values (or

* Corresponding author.
E-mail addresses: robert.bennetto.za@gmail.com (R. Bennetto), vuuren@sun.ac.za (J.H. Vuuren).

https://doi.org/10.1016/j.orp.2020.100177
Received 17 January 2020; Received in revised form 22 December 2020; Accepted 22 December 2020
Available online 24 December 2020

tuples) $\tau \in \mathbb{Z}^{|X(c)|}$ that satisfy c. $|X(c)|$ is the arity of c. Testing whether a tuple τ satisfies a constraint c is called a constraint check.

A constraint network within CP is defined as follows [4].

Definition 2. (*Constraint Network*) A constraint network is composed of:

- a finite sequence of integer variables $X = (x_1, \cdots, x_n)$,
- a domain for X, that is, a set $D = D(x_i) \times \ldots \times D(x_n)$, where $D(x_i) \subset \mathbb{Z}$ is the finite set of values that can be assumed by variable x_i, and
- a set of constraints $C = \{c_1, \ldots, c_e\}$, where variables in $X(c_j)$ are in X.

A network N is referred to in terms of its components pertaining to the variables, domains and constraints; that is, $N = (X,D,C)$. The process of searching requires that values are assigned to the variables of the network in some manner — typically a backtracking algorithm which ensures constraint adherence during the search process. The process of assigning a value to a variable is referred to as an *instantiation* of that variable [4].

Definition 3. (*Instantiation*) Given a network $N = (X, D, C)$,

- An *instantiation* V on $Y = (x_1, \ldots, x_k) \subset X$ is an assignment of values $v_1, \ldots, v_k$ to the variables $x_1, \ldots, x_k$, that is, V is a tuple on Y. V is denoted by $((x_1, v_1), \ldots, (x_k, v_k))$, where (x_i, v_i) denotes the value v_i for x_i.
- An instantiation V on Y is *valid* if, for all $x_i \in Y$, $V[x_i] \in D(x_i)$.
- An instantiation V on Y is *locally consistent* if and only if it is valid for all $c \in C$ with $X(c) \subset Y$ and $V[X(c)]$ satisfies c. If V is not locally consistent, it is *locally inconsistent*.
- A *solution* to a network N is an instantiation V on X which is locally consistent. The set of solutions of N is denoted by $sol(N)$.
- An instantiation V on Y is *globally consistent* (or *consistent*) if it can be extended to a solution (*i.e.* there exists an $s \in sol(N)$ with $V = s[Y]$).

A solution to a CSP is one where a valid instantiation is found for a given network N, for all variables X_N on domains D_N that satisfy all constraints C_N. A CSP specified as an optimisation problem will progressively add constraints, if a feasible solution is found, which restrict the next solution to have an objective function value superior to that of the previously found solution. In the case where the CP search completes an optimisation CSP with a feasible solution, an optimality proof is obtained. The flexibility of CSPs allows for a large range of discrete optimisation and satisfaction problems to be modelled.

CP employs backtracking search in order to find a valid instantiation for a CSP, or to demonstrate that no such instantiation exists, resulting in an unsatisfiable or infeasible network N. The words *instantiation* and *solution* may be used interchangeably in this context.

The branching strategy in CP can be described as a two stage process: Selecting which unassigned variable x_i in a network N should be assigned a value, followed by selecting a value from the domain $D(x_i)$ to assign to that variable. If all constraints are feasible with respect to the proposed assignment, the next unassigned variable is selected and the search continues by assigning a value or else backtracks if the assignment is infeasible. Popular techniques employed by commercial CP solvers include *impact-based search* (IBS) [15] and *failure-directed search* (FDS) [22] which both use statistics gathered during the search for each variable x_i to determine the ranking of a variable and value selection in the CP search tree. The performance of a particular branching strategy is not necessarily known *a priori* and as a result, portfolios of branching strategies are often used in practice.

2.2. Genetic programming

Given that the CP runtime may be large, it may be onerous to conduct a complete search. Many practitioners therefore use metaheuristics to find high-quality solutions if the problem domain is difficult to express as a CSP (such as in simulation environments) or if a good enough solution is sufficient. Numerous metaheuristics are available to practitioners for this purpose. The emphasis in this paper is on GP [11] which is a derivative of the *Genetic Algorithm* (GA) proposed by Goldberg [8]. The mechanics of the GP are similar to those of the GA in that they both employ a population-based approach and apply operators such as crossover, mutation and selection. An important difference between GAs and GPs is that the representation of the underlying chromosome in a GP takes the form of a tree-based structure as opposed to the vector-based structure commonly used in classic GAs. In the same way that a population member in a GA may contain multiple vector-based chromosomes, a GP member may also contain multiple tree structures, with *tree constraints* imposed to determine which nodes are permitted in a given tree. GP tree constraints are analogous to domain constraints on integer or real-based vector encodings employed in GAs. The operators used for crossover in GPs are designed to function on tree-based structures and bear little resemblance to their GA counterparts due to low independence between components of a tree should a modification be affected by crossover or mutation.

2.3. Multi-objective optimisation

Many real-world problems require measuring more than a single criterion when identifying the quality of a solution. One approach towards resolving this problem is to use a SAW of each of the objective terms and summarise this in a single, new, objective which can then be optimised in a classic single-objective paradigm. The technique of SAW has, however, received strong criticism [18] for several good reasons. SAW implies a known explicit trade-off between potentially conflicting objectives (*i.e.* being willing to forgo a units of dimension x for b units of dimension y). This requires knowledge of the domains of the dimensions being optimised and the user preferences between such dimensions (which may both be unknown). Secondly, there is an implied assumption of convexity when adopting SAW [5], which may not hold in practice and thus important portions of the search space may be inaccessible to a metaheuristic. This is not to say that SAW is not an appropriate method or that it is not useful in general — just that care should be taken in the treatment of the objective function components, understanding the utility function for each objective before a SAW is applied.

A question that naturally arises when working with multiple objectives is how one should compare one solution in terms of M objectives with another in terms of same objectives. The Pareto method is typically adopted to compare two candidate solutions, which results in multiple solutions being considered optimal. Many optimal solutions can be an uncomfortable proposition for an operations research practitioner, although Deb [5] suggests that a reasonable approach is to treat the problem of generating the Pareto frontier as a separate problem from selecting an optimal solution from the frontier. According to this two-step methodology in hand, one does not unnecessarily restrict the search space or bias the search process towards potentially poor-quality solutions.

In order to introduce the concept of Pareto-optimality, a definition of dominance with respect to a set of m objectives is adopted [5].

Definition 4. (*Multi-objective optimisation*) For set W, containing m objectives and a feasible set S of decision vectors, a multi-objective optimisation problem involves

$$\begin{array}{ll} \text{minimising} & f_m(\boldsymbol{x}), m \in W \\ \text{subject to} & \boldsymbol{x} \in S \end{array} \tag{1}$$

A dominance relation between two solutions $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ may be defined as follows:

Definition 5. (*Dominance*) A solution $\boldsymbol{x}^{(1)}$ dominates a solution $\boldsymbol{x}^{(2)}$ if both of the following conditions are met:

1. The solution $\boldsymbol{x}^{(1)}$ is no worse than $\boldsymbol{x}^{(2)}$ in all objectives, and
2. The solution $\boldsymbol{x}^{(1)}$ is strictly better than $\boldsymbol{x}(2)$ in at least one objective.

The notation $\boldsymbol{x}^{(1)} \preceq \boldsymbol{x}^{(2)}$ is adopted to indicate that solution $\boldsymbol{x}^{(2)}$ is dominated by solution $\boldsymbol{x}^{(1)}$ when both of these conditions are met. A situation may arise in which, given two solutions to (1), neither dominates the other, but both dominate other solutions. Solutions which are non-dominated can hence be separated from those which are dominated.

Definition 6. (*Non-dominated set*) Among a set of solutions P, the non-dominated set of solutions P' are those that are not dominated by any member of the the set P.

Definition 5 is referred to as the weak-domination criterion. Strong dominance can be defined follows:

Definition 7. (*Strong Dominance*) A solution $\boldsymbol{x}^{(1)}$ strongly dominates a solution $\boldsymbol{x}^{(2)}$ if $\boldsymbol{x}^{(1)}$ is strictly better than $\boldsymbol{x}^{(2)}$ in all m objectives.

The shorthand notation for strong dominance is $\boldsymbol{x}^{(1)} \prec \boldsymbol{x}^{(2)}$.

The principal change required to modify the widely used GA to cater for multiple objectives is in the selection of population members. The selection procedure in GAs requires comparing individuals in the population in order to determine which population members of high quality should be selected for breeding.

A highly successful approach was proposed by Deb et al. [6], called NSGA-II (Non-dominated Sorting Genetic Algorithm II). The algorithm not only addresses the computational complexity of other multi-objective GAs, such as that in [17] by the same authors, but is also capable of retaining diversity on the frontier without explicit parametrisation, ensuring that the Pareto frontier is adequately explored.

In order to ensure diversity in the selected population, Deb et al. [6] proposed a crowding distance calculation to bias selecting solutions which occur in lower density around the estimated Pareto Frontier. More formally, if two solutions have the equivalent rank as a result of non-dominated sorting algorithm, the solution with the lower crowding distance is selected. Deb et al. demonstrated that this algorithm significantly outperforms previous methods, such as SPEA [21] (strength--Pareto EA) and PAES [10] (Pareto-archived evolution strategy), when estimating the true Pareto frontier.

The next section contains a discussion on the methodology employed to use a multi-objective GP to determine suitable branching strategies for classes of CSPs, thereby extending the SAW approach adopted by Bennetto and Van Vuuren [3] to a multi-objective approach.

3. Methodology

The methodology proposed in this paper entails evolving a branching strategy obtained by GP which is evaluated in terms of several CSPs belonging to same problem class. A CSP instance i belonging to class m is denoted by $p_i^{(m)},\ i \in \{1,\ldots,n\}$ where n is the number of instances in the problem class. The set of problem classes is denoted by M. The shorthand $P^{(m)}$ is used to denote the collection of n CSP instances in class m. Branching strategy instance j developed for problem class m is denoted $S_j^{(m)}$. A description of the chromosomal representation for the branching strategy is provided in Section 3.1.

The GP is tasked with developing $S_j^{(m)}$, given a collection of training examples $P^{(m)}$ and a time budget of ten seconds to solve each $p_i^{(m)} \in P^{(m)}$. It is unlikely for CP to complete the solution for a non-trivial instance $p_i^{(m)}|S_j^{(m)}$ within this time budget and, as such, specific measures of the search performance are used to describe the efficacy of a candidate $S_j^{(m)}$. Aggregate measures of the search performance for $P^{(m)}|S_j^{(m)}$ are used to define the multi-objective function, described in Section 3.2.

The configuration parameters of the GP for each training run per-

Algorithm 1
Core procedure definitions.

```
Def CPSolve(p_i^(m), S_j^(m), T):
    set CSP instance p_i^(m);
    set Search Strategy S_j^(m);
    set Time limit T;
    F⃗ ← perform_search() ;
    return F⃗;
Def GPEval(P^(m), S_j^(m)):
    F⃗ ← 0⃗;
    foreach p_i^(m) ∈ P^(m) do
        F⃗ = F⃗+CPSolve(p_i^(m), S_j^(m), T = 10);
    end
    return F⃗;
Def GPSolve(P^(m)):
    set GP objective = GPEval;
    F ← NSGAII(params);
    return F;
Def Select(F, {a, b, c}):
    W ← F;
    W ← W[W_fa = min(W_fa)];
    W ← W[W_fb = min(W_fb)];
    W ← W[W_fc = min(W_fc)];
    return W_1;
```

Algorithm 2
Methodology pseudo code.

```
Q = ∅;
foreach m ∈ M^train do
    F ← GPSolve(P^(m));
    S_B^(m) ← Select(F, {2, 1, 3});
    Q[m] ← S_B^(m);
R = ∅;
foreach m ∈ M do
    foreach p_i^(m) ∈ P^(m) do
        R[p_i^(m)] ← CPSolve(p_i^(m), Q[m], T = 1200)
foreach m ∈ M do
    foreach p̄_i^(m) ∈ P̄^(m) do
        R[p_i^(m)] ← CPSolve(p̄_i^(m), Q[m], T = 1200)
tabulate(R)
```

Table 1
Description of the node and terminal set used in Tree 1, the variable selector.

Function Name	Type	Arity	Description	Abbreviation
Add	Node	2	Simple addition	+
Mul	Node	2	Simple multiplication	*
Inv	Node	1	Protected inversion	Inv
Neg	Node	1	Negation	-
Rand	Terminal	0	Random uniform number	rand
MinX	Terminal	0	The minimum feasible value in $D(x_i)$	Min(x)
MaxX	Terminal	0	The maximum feasible value in $D(x_i)$	Max(x)
SizeX	Terminal	0	The cardinality of feasible domain $D(x_i)$	Size(x)
SuccessRateX	Terminal	0	The success rate of value assignments to variable x_i	SuccRate(x)
ImpactX	Terminal	0	The (sum of) impact of variable x_i	Impt(x)

taining to class m is described in Section 3.3. A description of the test data suite and default search strategy is provided in Section 3.4. Once the GP has reached the termination criterion, a "best" strategy $S_B^{(m)}$ is selected from the approximate Pareto frontier. This is done using a hierarchical selection scheme described in Section 3.5. $S_B^{(m)}$ is then evaluated on the training samples $P^{(m)}$, given a full 20 min to complete the search. This complete evaluation on the training data allows for a comparison between the performance of $S_B^{(m)}$ and the default search strategy employed by the solver, denoted by S_D. Note that the default search strategy does not differentiate by problem class.

As a final step, $S_B^{(m)}$ is evaluated on unseen CSP data belonging to the same problem class, denoted by $\overline{P}^{(m)}$. The evaluation of $\overline{P}^{(m)}|S_B^{(m)}$ is compared to S_D in Section 4. A summary of the key procedures outlined in the methodology is provided in Algorithm 1 and the pseudo code outlining the process-flow of the complete train and test methodology is provided in Algorithm 2.

3.1. GP representation

A strategy $S_j^{(m)}$ comprises four components. A variable selector tree and a value selector tree are employed, as well as two single terminal node decisions encapsulating the parameters for whether *nogoods* should be used by the search and the *log restart size*. This enables the GP so explore not only the explicit branching strategy, but also the interaction with higher level parameters which may also be useful to determine an effective search strategy. The variable and value selector tree components are described in Tables 1 and 2, respectively. The *nogoods* parameter is a binary variable and the *log restart size* an integer variable on the range [0,17]. These four components implicitly define the resulting strategy search space for the GP.

The function set in Table 1 allows for the creation of two popular search methods through the use of SuccessRate and ImpactX, namely FDS and IBS. Other search heuristics commonly used in CP, such as Dom, can be generated through the use of Size(X). It would thus also be possible to create the inverse strategies for IBS and FDS by combining the Inv operator with the associated operator.

The GP tree functions presented in Tables 1 and 2 would typically be described as an arithmetic set and are used during the CP search to create a ranking value for each variable and value assignment tuple. The unassigned variable with the lowest rank is selected as the branching variable, after which the value for the selected variable with the lowest rank is selected as the next value assignment attempt during the CP search.

3.2. GP objective function

The GP objective function is created by aggregating three CP-search performance measures. The first is the maximum search depth achieved for $p_i^{(m)}|S_j^{(m)}$ where the total number of variables and the maximum search depth are denoted by V_i and d_i, respectively, for $p_i^{(m)}$. The maximum search depth is stated as a percentage of the total number of CSP variable and is given by

$$f_1(P^{(m)}) = \sum_{i \in P^{(m)}} \frac{V_i - d_i}{V_i}, \tag{2}$$

which is thus normalised to the interval [0,1], where a zero value indicates that a feasible solution has been found by the search procedure.

When an optimisation objective is defined, the quality of the best objective value o_i obtained for $p_i^{(m)}$ is measured by the function

$$f_2(P^{(m)}) = \sum_{i \in P^{(m)}} \frac{f(o_i)}{o_i^{\max} - o_i^{\min}}, \tag{3}$$

where

$$f(o_i) = \begin{cases} o_i^{\max} - o_i, & \text{if the sense is maximisation} \\ o_i - o_i^{\min}, & \text{otherwise.} \end{cases} \tag{4}$$

The domain of the variable o_i is given by $[o_i^{\min}, o_i^{\max}]$, where the values of $o_i^{\min}$ and $o_i^{\max}$ are determined after initial propagation (a deterministic procedure) has been performed. This ensures that the range $[o_i^{\min}, o_i^{\max}]$ is reasonably well bounded. The objective *sense* is given by the direction of the CSP objective function, which is either minimisation or maximisation. If a CP search results in no feasible incumbent being found, the value of (3) is set to one. The value of (3) is set to zero for CSPs which do not define an objective function.

The third objective function component

$$f_3(P^{(m)}) = \sum_{i \in P^{(m)}} \frac{t_i}{T}. \tag{5}$$

measures as a function of the time taken to solve $p_i^{(m)}$, where the maximum time is denoted by T and the time expended by the search is given by t_i.

Table 2
Description of the node and terminal set used in Tree 2, the value selector.

Function Name	Type	Arity	Description	Abbreviation
Add	Node	2	Simple addition	+
Mul	Node	2	Simple multiplication	*
Inv	Node	1	Protected inversion	Inv
Neg	Node	1	Negation	-
Rand	Terminal	0	Random number (uniform)	rand
LocalImpactValueX	Terminal	0	The local impact of the assignment of value v to variable x_i	Limpt(x, v)
NumberOfFailsValueX	Terminal	0	The number of fails of the assignment of value v to variable x_i	Fails(x, v)
NumberOfInstantiationsValX	Terminal	0	The number of instantiations of value v to variable x_i	Inst(x, v)

The definition of f_3 in (5) normalises the values obtained to the same domain as the functions in (2) and (3). A SAW objective function is written as

$$f_{saw} = f_1 + f_2 + f_3. \tag{6}$$

It would be reasonable to question the usage of a unit weight for each of the terms used in the formation of a SAW objective (6). Due to the correlation structure between these individual objectives, however, it would seem unnecessary to specify a more complex weighting scheme. Domains are orientated in functions (2), (3) and (5) such that good solutions are close to zero and poor solutions near one. In a SAW scheme, the sum of these objective components also correlate to the extremes of the resulting aggregate domain. A problem arises in the consideration of the algorithmic progress between these domain extremes, where subtle trade-offs between objectives may be required.

A SAW scheme weights the importance of feasibility (2) and the expended time (5) equally. One may consider an example of a satisfiability CSP where some subset of instances in $P^{(m)}$ are feasible for strategy $S_i^{(m)}$ (completing with small values for the expressions in (2) and (5)) while others are infeasible (with poor values for the term in (2)). $S_i^{(m)}$ may be compared with a alternate strategy $S_j^{(m)}$, which produces no feasible solutions in $P^{(m)}$ but is generally close to achieving a feasible outcome over instances evaluated in $P^{(m)}|S_j^{(m)}$ such that f_{saw} may be greater then the evaluation of f_{saw} for $P^{(m)}|S_i^{(m)}$. This means that $S_i^{(m)}$ would be preferred over $S_j^{(m)}$, which may not be the desired outcome. It may be found that, with a sufficient time budget, the strategy of $S_j^{(m)}$ produces a smaller value for f_{saw}. The argument here can me made in either direction simply by altering the degree of feasibility achieved for the term (2) in f_{saw}. This example illustrates the potential bias towards different search strategies based on a SAW scheme.

It would be possible to mimic a tiered hierarchical objective function by applying weights and offsets in the construction of a SAW objective such that the point at which a trade-off between objectives becomes negligible, with the emphasis being placed on first minimising (2), then (3) and so forth. One may argue that this may decrease population diversity in the evolutionary algorithm as it creates a myopic objective function in which a large emphasis is then placed on each successive objective function term. There is a risk that some strategies which may be exploring features of the local search space are disregarded during earlier selection iterations as they are aggressively dominated (in cost) by a single strategy that performs well during initial generations of the GP. There are several works which highlight the importance of having population diversity in initial populations that are similar in cost so as to avoid pre-mature convergence in GAs.

It is prudent to note that there are no assurances as to the convexity of the objective components (2), (3) and (5) in a SAW scheme, which implies that no linear combination of these terms is capable of expressing the true Pareto frontier, resulting in portions of the optimal frontier being disregarded as being inferior. It is a natural extension rather to consider a multi-objective approach which can lead to more favourable transitions of candidate solutions to an optimal policy. The multi-objective version of the problem is thus to

$$\text{minimise } f_k(P^{(m)}), k \in \{1,2,3\}. \tag{7}$$

3.3. GP configuration

The multi-objective variation of the GP employs the same parameters for crossover and mutation as those used by Bennetto and Van Vuuren [3], but adopts the NSGA-II selection algorithm to maintain the Pareto frontier of best solutions found. The ability of the NSGA-II algorithm to maintain a frontier of best solutions requires that the population size be adjusted. Frontier solutions are carried forward to the next generation in the GP and it is desirable to have a similar number of individual evaluations when performing comparisons with the SAW GP. During experimentation, it was found that a population size of 150 individuals produced a close match to the target number of individual evaluations in the SAW GP run. A summary of the parameters of the GP run for multi-objective optimisation is provided in Table 3.

Table 3
GP parameters for multi-objective optimisation.

Parameter	Value	Description
Generations	15	The number of population iterations performed by the GP.
Population size	150	The size of the population at each iteration (including the Pareto frontier).
Elitism	No	Elitism is indirectly maintained through the frontier being persisted from one iteration to another and the idea of an explicit "best" individual in multi-objective optimisation is not possible.
Tournament size	None	Selection is managed through the non-dominated sorting algorithm.
Crossover rate	0.9	The probability that a node in the tree is selected for a crossover operation. Crossover exchanges the two resulting subtrees between two individuals at compatible nodes.
Maximum crossover depth	17	The limit on the selection of nodes for crossover.
Mutation rate	0.1	The rate at which subtree mutation is applied to tree nodes.
Mutation type	Subtree	Subtree mutation invokes a call to the *Grow* method which randomly creates a new subtree of depth 5.
Non-terminal Selection Probability	0.9	When generating new trees, the probability that a node with non-zero arity is selected at random.
Terminal Selection Rate	0.1	When generating new trees, the probability that a terminal is selected at random.
Seeding mechanism	50/50 Grow/Full	Also referred to as the half-builder. Half the population is seeded using the *Grow* method and the other half through the *Full* method.
Full depth range	[2,6]	The target range of minimum and maximum depths for a tree created using the *Full* method.
Grow depth range	[5,5]	The target range of minimum and maximum depths for a tree created using the *Grow* method.

Table 4
Constraint satisfaction problem status codes.

Code	Description
S	A feasible **S**olution was found.
C	The search was **C**ompleted.
SC	A **S**olution was found and the search was **C**ompleted.
UNK	No solution was found within time limit, invalid solution or out-of-memory error.

The Java-based evolutionary framework ECJ [12] implementation is used to implement the grammar presented in Tables 1 and 2 as well as the configurations specified in Table 3.

3.4. Test data and CP solver

The 2015 MiniZinc challenge problems [19] were previously used by Bennetto and Van Vuuren [3] to benchmark the performance of the GP to develop branching strategies. The same nineteen problem instances are used here to test the multi-objective variant of the GP search.

The result of a particular CSP search is assigned a status code in the MiniZinc challenge. A summary of the status codes are shown in Table 4. An ideal status code for optimisation problems is 'SC' (completed with optimality proof) and either 'S' or 'C' for pure satisfiability problems, where the codes correspond to a feasibility proof and an infeasibility

Table 5
Summary of Pareto frontier selection scheme results for multi-objective GP across all training set problems.

	Full Evaluation Results				Sampling limit terms		
Selection Scheme	# S	# SC	# C	# UNK	$\sum f_1$	$\sum f_2$	$\sum f_3$
$\{1,2,3\}$	56	29	0	10	1.35	35.03	60.51
$\{1,3,2\}$	58	27	0	10	1.35	36.10	60.40
$\{2,1,3\}$	57	29	0	9	1.35	33.66	60.41
$\{2,3,1\}$	56	29	0	10	1.62	34.66	61.10
$\{3,1,2\}$	56	27	0	12	1.92	38.10	61.97
$\{3,2,1\}$	57	27	0	11	1.90	36.73	61.97
f_{saw}	56	29	0	10	1.62	35.16	61.11

proof, respectively.

The *Google Or-Tools* (ORT) solver [14] is employed as a reference point from which to measure high-quality search strategies found as the ORT solver was the top performing open-source solver in the free-search category of the 2015 MiniZinc Challenge. The ORT default strategy employed is a follows a series of heuristic dives and IBS. The maximum numbers of heuristic dives are limited and consist of common heuristics in CP, such as selecting the smallest or largest variable domain in conjunction with the smallest or largest feasible value. A single heuristic dive is limited by the number of branches that may be explored. Heuristic variable domain splits are used in addition to the dives and IBS which are isolated to variables with larger domains. IBS is the default search mechanism once heuristics have completed in ORT.

The training data consists of five problem instances per problem class. The performance of a candidate branching strategy is evaluated over all five instances, with 10 s of search time permitted per problem instance $p_i^{(m)}$. The objective function evaluation time for an individual in the GP population is typically less than 60 s, as a small amount of additional time is required to compile the strategies which are embedded in the CP search procedure. The ORT solver is given 20 min of search time employing the default search algorithm, serving as a benchmark with which to compare the performance of the final evolved GP solutions, per problem class, which are also then given 20 min of search time. The best solutions previously found by a SAW scheme [3] are included and labelled GP SAW. Multi-objective GP results are denoted by the shorthand 'GP MO' in the tables of results.

The total computational budget of the GP is considerably larger than the total time expended by the ORT default search algorithm. One may, however, consider this as an offline training procedure whose purpose is to evolve a search strategy which may yield improved performance on unseen problem instances of the same class.

The empirical results were performed using the same hardware specification as in the SAW experiments previously conducted by Bennetto and Van Vuuren [3] in order to support a valid comparison between the results obtained.

3.5. Strategy selection

As mentioned, Deb [5] suggested that the processes of producing a Pareto frontier and selecting a solution from the frontier be treated as independent steps. This section contains a description of the methodology adopted to evaluate different schemes in order to determine a single $S_B^{(m)}$ from the generated Pareto frontier.

The goal of the GP training runs is to produce $S_B^{(m)}$ for a problem class upon expending a small amount of search time. In a classic single-objective GP, $S_B^{(m)}$ is simply the strategy with the lowest cost for f_{saw} (6). The multi-objective GP, on the other hand, returns the Pareto frontier of best strategies found. It is therefore required to determine the criteria by which a single individual may be selected from the Pareto frontier to use as the candidate search strategy for a full evaluation. $S_B^{(m)}$ may be selected from the frontier based on f_{saw} value, some other weighting scheme, or even hierarchically.

Each $S_j^{(m)}$ on the Pareto frontier which would qualify as a candidate $S_B^{(m)}$ according to either a hierarchical selection scheme or f_{saw}, are evaluated by executing a 20 min search (as opposed to a 10 s search permitted during training). The evaluation of the selection schemes for the Pareto frontier are carried out strictly on measurements available to the GP search based on the limited search time for a candidate $S_j^{(m)}$. By restricting the selection criteria to measurements available at the end of the GP search, a single strategy is selected from each frontier without having to evaluate the final quality of all individuals on the frontier in future experiments. Six hierarchical selection schemes are provided in Table 5 which rank the objective components in their order of importance.

The difference in overall outcomes in Table 5 is determined by a subset of problems for which multiple solutions are available on the Pareto frontier (the set of blue points in Fig. 1). It is interesting to note that recomputing the results of a multi-objective search strategy using f_{saw} in (6) as a selection criterion produces a relatively average overall result in the context of other selection schemes.

The scheme $\{2,1,3\}$ is considered the preferred scheme for comparing results as not only does it achieve a large number of feasibility proofs, but also the least number of unknown status codes. The overall sum of normalised objective function values for this selection scheme is also the smallest among the set of comparisons. This scheme selects a point from the Pareto frontier hierarchically, first according to the smallest normalised objective sum (3), then according to the normalised remaining minimum infeasibility (2) and finally according to the normalised solution time (5).

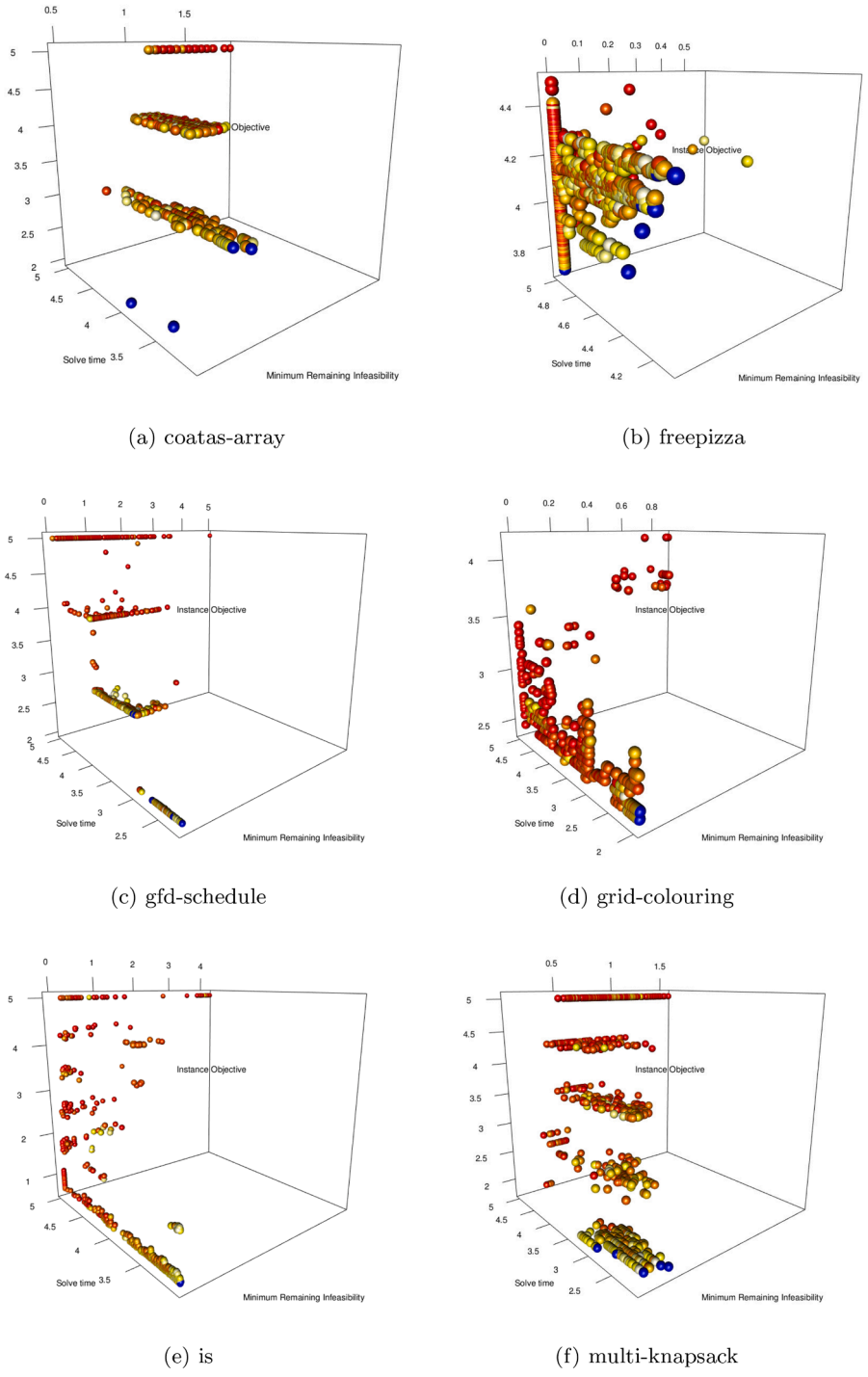
4. Results

A subset of the multi-objective GP runs is provided in Fig. 1. The plots are provided in three dimensions where all strategies are able to produce variation in all components of the objective function (7). The latter plots are provided in two dimensions as there was no variation in the omitted component values. Lastly, problem classes p1f and triangular were omitted from the plots as no strategies were able to produce variation in more than a single objective function component within the sampling limit[1]. Evaluated solutions which are concluded to lie on the Pareto frontier are coloured in blue in Fig. 1. Solutions not on this frontier are coloured based on a heat scale indicating at which generation during the GP search the individuals were created, red and yellow being used for individuals created near the start and the end of the GP run, respectively.

The mapping problem class in Fig. 1 illustrates the correlation structure between the objective function components in that small values of remaining infeasibility are associated with small objective instance values. The strength of this correlation varies between problem instances, which is most prevalent in the zephyrus problem class. The tdtsp problem class clearly illustrates the convergence progression of the GP as the majority of solutions are of poor quality at the start of the GP run with high-quality solutions being formed at the end of the run (light yellow). The Pareto frontiers vary in size from a single dominant solution (is) to several optimal solutions (freepizza).

The complete set of all runs comparing the results between the default ORT search, the single-objective GP using a SAW scheme and the multi-objective GP using the $\{2,1,3\}$ scheme to select the preferred strategy from the Pareto frontier is provided in Table 6. The difference in

[1] In the case of p1f this raises the question as to whether the sampling limit should be increased for larger problems in order to evolve meaningful strategies. The triangular problem class has a tendency always to admit a feasible solution, and utilise the full time limit, leaving only the instance objective function as a facet.



(a) coatas-array (b) freepizza

(c) gfd-schedule (d) grid-colouring

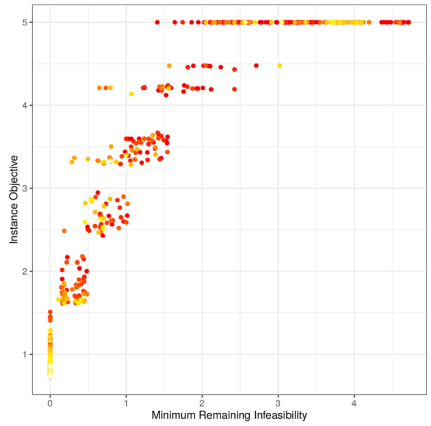
(e) is (f) multi-knapsack

Fig. 1. GP run results using NSGA-II for a subset of problem classes (continued).

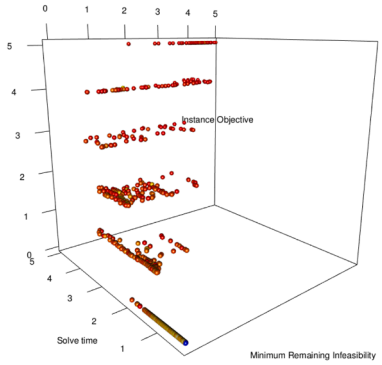
search performance is shown in the columns marked Δ, with differences being indicated as either improvements (+) or deteriorations (-) in the respective measurements for the single-objective and multi-objective searches relative to the default ORT search algorithm. In addition, the differences in performance between the SAW and MO GP schemes are provided in the last three columns.

Scoring the differences in objective function values obtained by the CP search for optimisation problems in Table 6 is only performed when both objective values are present. The reason for this is that a penalty has already been accounted for in the difference between status codes, *i.e.* if one strategy produces S and another UNK, there will only be one objective function value (corresponding to the S code). Thus the impact of a superior or inferior search has already been encapsulated in the status code or search time in such instances. Differences between search times of less than one second are not considered sufficiently significant to contribute to the scoring employed.

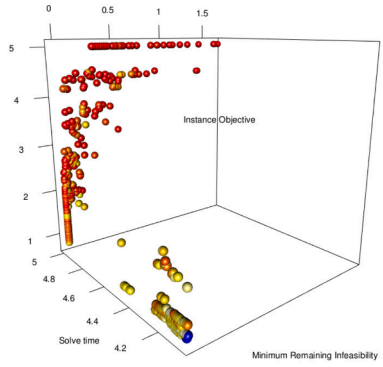
The summary of results in Table 7 provides an overview of where significant improvements could be made in the instance objective and time to solve. It was previously established [3] that the GP SAW scheme significantly outperforms the default ORT strategy at the 5% level of



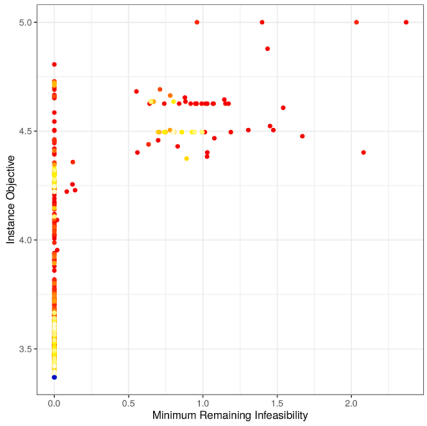
(g) mapping



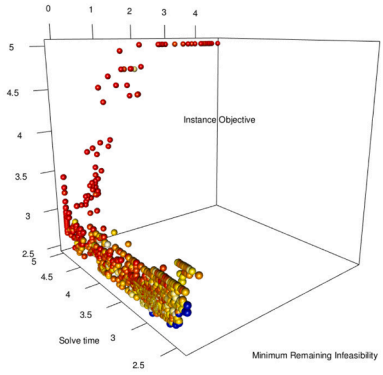
(h) nmseq



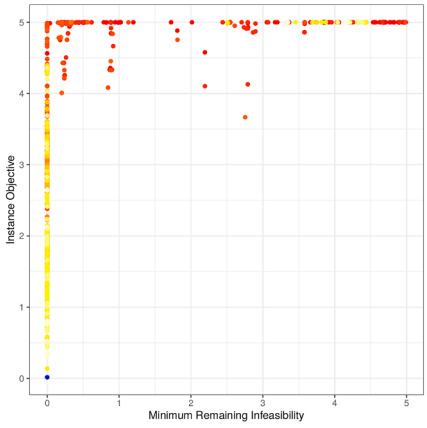
(i) project-planning



(j) spot5



(k) tdtsp



(l) zephyrus

Fig. 1. (*continued*).

significance. Table 8 summarises the *p*-values obtained for the multi-objective results which also significantly outperform the default ORT search at the 5% level of significance. The significance test employed is the non-parametric signed Wilcoxon rank test [20] which tests for significant differences in matched samples. The Wilcoxon test makes no distributional assumptions about the mean and is a strong indicator of significant differences if the null hypothesis is rejected.

The Wilcoxon rank test also produces a significant *p*-value (0.02435) at the 5% significance level when testing for a difference in performance between the SAW and MO GP schemes at an aggregate level. The majority of the positive contributions to performance are as a result of strategies found which result in an improved best objective function value found for the target CSP, namely project-planning (+ 5), freepizza (+ 4) and is (+ 3).

The multi-objective scheme was able to find some interesting strategies which outperformed the SAW scheme. The selected costas-array search strategy performed slightly worse on the first two problem instances (by a total of 11 s) but was able to provide a solution to problem

Table 6
ORT, single-objective and multi-objective GP full search comparison (training set).

					ORT			GP SAW (Δ ORT)						GP MO (Δ ORT)						GP MO (Δ GP SAW)		
Problem Class	Instance	Sense	Variables	Constraints	Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ	Code	Time	Best
	16		16	802	S	19		S		1	+			S		6	+				−	
	17		17	954	S	240		S		1	+			S		7	+				−	
costas-array	18	None	18	1 124	S	17		S		364	-			S		8	+				+	
	19		19	1 313	UNK	1 200		UNK		1 200				S	+	105	+			+	+	
	20		20	1 522	UNK	1 200		UNK		1 200				UNK		1 200						
	A-n37-k5.vrp		610	23 002	S	1 200	1 642	S		1 200		1 832	−	S		1 200		2 122	−			−
	A-n64-k9.vrp		1 069	69 172	S	1 200	3 486	S		1 200		3 545	−	S		1 200		4 046	−			−
cvrp	B-n45-k5.vrp	Min	746	34 098	S	1 200	2 728	S		1 200		2 628	+	S		1 200		2 467	+			+
	P-n16-k8.vrp		253	4 228	S	1 200	502	S		1 200		502		S		1 200		450	+			+
	simple2		117	1 020	SC	377	34	SC		30	+	34		SC		19	+	34			+	
	pizza27		180	32 600	S	1 200	882 425	S		1 200		822 299	+	S		1 200		761 294	+			+
	pizza39		190	36 890	S	1 200	939 352	S		1 200		987 968	−	S		1 200		837 068	+			+
freepizza	pizza45	Min	140	19 759	S	1 200	656 489	S		1 200		641 397	+	S		1 200		571 934	+			+
	pizza6		10	159	SC	610	210	SC		5	+	210		SC		113	+	210			−	
	pizza78		200	40 229	S	1 200	901 717	S		1 200		896 600	+	S		1 200		714 755	+			+
	n120f5d50m50k20		7 816	19 629	S	1 200	19 463	SC	+	1	+	1	+	SC	+	3	+	1	+		−	
gfd-schedule	n180f7d50m30k18	Min	17 186	45 699	SC	1	1	UNK	−	1 200	−			UNK	−	1 200	−					
	n30f3d30m7k4		616	1 374	UNK	1 200		SC	+	1	+	1		SC	+	1	+	1				
	n50f7d40m10k4		1 540	3 660	UNK	1 200		SC	+	1	+	1		SC	+	2	+	1				
	n75f5d30m20k20		3 129	6 782	UNK	1 200		SC	+	1	+	1		UNK		1 200				−	−	
	10_5		51	376	SC	31	3	SC		1	+	3		SC		1	+	3				
	13_11		144	1 717	S	1 200	7	S		1 200		5	+	S		1 200		4	+			+
grid-colouring	19_17	Min	324	5 815	S	1 200	12	S		1 200		7	+	S		1 200		5	+			+
	4_11		45	331	S	1 200	4	SC	+	1	+	3	+	SC	+	1	+	3	+			
	4_8		33	193	SC	2	3	SC		1		3		SC		1		3				
	1YHXeG1xYs		913	1 921	S	1 200	194 048	S		1 200		145 440	+	S		1 200		99 328	+			+
	A3PZaPjnUz		507	926	S	1 200	144 896	SC	+	1	+	103 936	+	SC	+	1	+	103 936	+			
is	HgSWGJHxY5	Min	835	1 680	S	1 200	251 200	S		1 200		276 000	−	SC	+	1 117	+	102 176	+	+	+	+
	jZ9pQqRxJ2		508	799	SC	82	210 944	SC		2	+	210 944		SC		1	+	210 944				
	y21PnVA2Hj		860	1 840	S	1 200	236 544	S		1 200		165 776	+	S		1 200		127 088	+			+
	full2x2		172	235	S	1 200	1 103	S		1 200		801	+	S		1 200		795	+			+
	mesh2x2_mpeg		522	803	S	1 200	726	S		1 200		1 436	−	S		1 200		1 116	−			+
mapping	mesh3x3_2	Min	348	497	UNK	1 200		S	+	1 200		1 631		S	+	1 200		1 623				+
	mesh3x3_mpeg_2		1 302	1 709	S	1 200	2 197	S		1 200		1 188	+	S		1 200		1 211	+			−
	ring_2		300	473	S	1 200	2 090	S		1 200		1 940	+	S		1 200		1 940	+			
	mknap1-6		50	8	UNK	1 200		SC	+	7	+	16 537		SC	+	7	+	16 537				
	mknap2-1		60	33	SC	158	7 772	SC		1	+	7 772		SC		2	+	7 772				

(*continued on next page*)

Table 6 (*continued*)

					ORT			GP SAW (Δ ORT)						GP MO (Δ ORT)						GP MO (Δ GP SAW)			
multi-knapsack	mknap2-2	Max	60	33	S	1 200	8 722	SC	+	8	+	8 722		SC	+	8	+	8 722					
	mknap2-20		50	8	SC	3	6 339	SC		1	+	6 339		SC		1	+	6 339					
	mknap2-32		80	8	UNK	1 200		S	+	1 200		8 947		SC	+	547	+	8 947		+	+		
	176		177	530	S	6		S		1	+			S		1	+						
	207		208	623	S	7		S		1	+			S		1	+						
nmseq	269	None	270	809	S	45		S		2	+			S		2	+						
	393		394	1 181	S	244		S		2	+			S		4	+				−		
	83		84	251	S	1		S		1				S		1							
	flener_et_al_10_350_100		15 349	28 405	S	1 200	65	UNK	−	1 200				UNK	−	1 200							
	medium_10_100_30		4 349	8 155	S	1 200	13	S		1 200		21	−	S		1 200		10	+				+
opd	small_bibd_10_30_09	Min	1 269	2 485	SC	1 107	2	S	−	1 200	−	3	−	S	−	1 200	−	3	−				
	small_bibd_11_22_10		1 021	2 161	S	1 200	5	S		1 200		5		S		1 200		5					
	small_bibd_13_26_06		1 465	3 385	SC	1 078	1	S	−	1 200	−	2	−	S	−	1 200	−	2	−				
						ORT				GP SAW (Δ ORT)						GP MO (Δ ORT)						GP MO (Δ GP SAW)	
Problem Class	Instance	Sense	Variables	Constraints	Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ	Code	Time	Best	
	problem_20_20_1		744	1 668	S	1 200	11	S		1 200		11		S		1 200		11					
	problem_30_15_1		783	1 689	SC	15	14	S	−	1 200	−	14		S	−	1 200	−	14					
open_stacks	wbo_10_20_1	Min	379	949	SC	303	5	S	−	1 200	−	5		S	−	1 200	−	5					
	wbop_15_30_1		899	2 205	S	1 200	7	S		1 200		7		S		1 200		6	+				+
	wbp_20_20_1		739	1 606	S	1 200	4	S		1 200		4		S		1 200		4					
	12		1 617	12 309	S	1 200	602	UNK	−	1 200				UNK	−	1 200							
	13		2 070	16 812	C	20		UNK	−	1 200	−			UNK	−	1 200	−						
p1f	14	Min	2 600	22 438	S	1 200	1 008	UNK	−	1 200				UNK	−	1 200							
	15		3 213	29 358	C	68		UNK	−	1 200	−			UNK	−	1 200	−						
	17		4 712	47 824	UNK	1 200		UNK		1 200				UNK		1 200							
	ProjectPlannertest_12_7		3 463	693	S	1 200	63	S		1 200		19	+	SC	+	1	+	17	+	+	+	+	
	ProjectPlannertest_14_7		12 801	920	S	1 200	78	S		1 200		32	+	SC	+	347	+	27	+	+	+	+	
project-planning	ProjectPlannertest_15_6	Min	25 156	1 050	S	1 200	66	S		1 200		37	+	SC	+	994	+	31	+	+	+	+	
	ProjectPlannertest_16_6		49 803	1 178	S	1 200	39	S		1 200		35	+	S		1 200		31	+				+
	ProjectPlannertest_16_8		49 803	1 180	S	1 200	39	S		1 200		35	+	S		1 200		31	+				+
	i14-9		2 467	2 051	SC	252	6 513	S	−	1 200	−	6 720	−	S	−	1 200	−	6 526	−				+
	i6-11		544	495	SC	207	895	SC		9	+	895		S	−	1 200	−	896	−	−	−	−	−
radiation	i6-21	Min	1 036	919	SC	143	1 413	S	−	1 200	−	1 718	−	S	−	1 200	−	1 417	−				+
	i7-9		619	548	SC	7	1 007	SC		6		1 007		S	−	1 200	−	1 009	−	−	−	−	−
	i9-11		1 265	1 066	SC	427	2 141	SC		123	+	2 141		S	−	1 200	−	2 151	−	−	−	−	−
	chicroster_dataset_11		559	564	SC	1	17	SC		1		17		SC		1		17					
	chicroster_dataset_17		671	676	SC	1	17	SC		1		17		SC		1		17					
roster	chicroster_dataset_2	Min	189	248	SC	0	0	SC		1		0		SC		1		0					
	chicroster_dataset_5		279	294	SC	1	6	SC		1		6		SC		1		6					
	chicroster_dataset_7		323	363	SC	1	0	SC		1		0		SC		1		0					
	1401		10 964	24 078	S	1 200	521 097	S		1 200		496 114	+	S		1 200		500 112	+				−
spot5	28	Min	5 227	10 642	S	1 200	284 158	S		1 200		276 105	+	S		1 200		277 105	+				−
	414		10 109	24 373	S	1 200	42 564	S		1 200		44 501	−	S		1 200		49 510	−				−
	503		636	1 130	S	1 200	15 177	S		1 200		11 125	+	S		1 200		11 134	+				−
	54		272	462	S	1 200	81	S		1 200		37		S		1 200		37					

(*continued on next page*)

Table 6 (*continued*)

					ORT		GP SAW (Δ ORT)				GP MO (Δ ORT)				GP MO (Δ GP SAW)			
	inst_10_24_10	Min	67	358	S	1 200	13 917	SC	+	5	+	9 192	SC	+	3	+	9 192	+
	inst_10_34_00		67	358	S	1 200	8 353	SC	+	3	+	6 662	SC	+	2	+	6 662	+
tdtsp	inst_10_42_10	Min	67	358	S	1 200	15 329	SC	+	4	+	8 486	SC	+	2	+	8 486	+
	inst_20_14_10		137	1 118	S	1 200	17 449	S		1 200	+	14 889	S		1 200		15 801	+
	inst_20_25_00		137	1 118	S	1 200	19 898	S		1 200	+	16 655	S		1 200		15 945	+
	n10		55	2	SC	89	20	SC		159		20	S		1 200		20	
	n16		136	2	S	1 200	35	S		1 200		34	S		1 200		34	
triangular	n22	Max	253	2	S	1 200	48	S		1 200	+	50	S		1 200		50	+
	n28		406	2	S	1 200	61	S		1 200	+	64	S		1 200		65	+
	n37		703	2	S	1 200	80	S		1 200	+	87	S		1 200		88	+
	zephyrus_15_10		462	995	S	1 200	36	S		1 200		36	S		1 200		36	
	zephyrus_20_20		612	1 320	S	1 200	66	S		1 200		66	S		1 200		66	
zephyrus	zephyrus_5_20	Min	162	345	S	1 200	66	SC	+	830	+	66	SC	+	416	+	66	+
	zephyrus_5_4		162	345	S	1 200	18	S		1 200		18	SC	+	390	+	18	+
	zephyrus-FH-2-15		461	995	SC	237	12	S	–	1 200	–	12	S	–	1 200	–	12	+

Table 7
GP SAW and multi-objective results summary by problem class (training set).

	GP SAW (Δ ORT)			GP MO (Δ ORT)			GP MO (Δ GP SAW)		
Problem Class	−	+	Δ	−	+	Δ	−	+	Δ
costas-array	1	2	1	0	5	5	2	3	1
cvrp	2	2	0	2	3	1	2	3	1
freepizza	1	4	3	0	5	5	1	4	3
gfd-schedule	2	9	7	2	7	5	3	0	− 3
grid-colouring	0	6	6	0	6	6	0	2	2
is	1	6	5	0	9	9	0	5	5
mapping	1	4	3	1	4	3	1	3	2
multi-knapsack	0	7	7	0	8	8	0	2	2
nmseq	0	4	4	0	4	4	1	0	− 1
opd	8	0	− 8	7	1	− 6	0	1	1
open_stacks	4	0	− 4	4	1	− 3	0	1	1
p1f	6	0	− 6	6	0	− 6	0	0	0
project-planning	0	5	5	0	11	11	0	11	11
radiation	6	2	− 4	15	0	− 15	9	2	− 7
spot5	1	4	3	1	4	3	4	0	− 4
tdtsp	0	11	11	0	11	11	1	3	2
triangular	2	3	1	3	3	0	2	2	0
zephyrus	2	2	0	2	4	2	0	3	3
Total	37	71	34	43	86	43	26	45	19

instance 19 as well as a solution in less time for problem instance 18. The strategies selected for the freepizza, is and project-planning problem classes all provided improvements in the objective function values obtained within the time limit. The strategy for the project-planning problem class also obtained optimality proofs for three of the problem instances.

As all parameters (where possible) were held constant between the SAW and multi-objective GA, the improvements obtained by applying multi-objective search over the SAW scheme are attributed to the NSGA-II selection algorithm employed. The non-dominated sorting algorithm replaces the tournament selection procedure previously employed — the result of which is that multiple solutions may be considered of highest quality during a given iteration in the GA. Since the multi-objective GA always carries forward the current Pareto frontier of solutions, it is possible that additional genetic diversity is maintained in the population for longer which may have been lost earlier on as a result of the tournament selection operator. The other possibility is that high-quality solutions (on the Pareto frontier) are more likely to be combined in a crossover operator resulting in solutions which measurably address different objective function components in an effective manner.

4.1. *Extendibility to unseen problem instances*

The search strategies selected during the training process were evaluated on unseen problem instances that are labelled as being from the same class of problem. New problem instances were drawn from the CSPlib [9] for testing consistent with the test set used by [3]. The detailed results are provided in Table 9.

A summary of the results are provided in Table 10. It is noteworthy that neither the SAW scheme nor multi-objective GP scheme was able to construct search strategies for the opd, open_stacks, p1f and radiation problem classes which are competitive with the ORT default search strategy and were thus not reported in the test set results. The expectation that poorly performing strategies on the training data are carried forward to the test set was found to be true — while this does suggest that it is non-trivial to perform well, it can also be seen in the evaluation results of the GP in Fig. 1. The zephyrus problem instances were added to the test set as the multi-objective GP was able to construct a strategy for this problem class which outperformed the ORT search strategy and the single-objective GP on the training data. Test data for costas-array were also included, but it is clear that the smaller instances are largely trivial to solve.

Table 8
Wilcoxon rank test results for signficiant differences to a mean of zero in improvements and deteriorations by measurement category and overall changes. An asterisk indicates significant *p*-values at the 5% level.

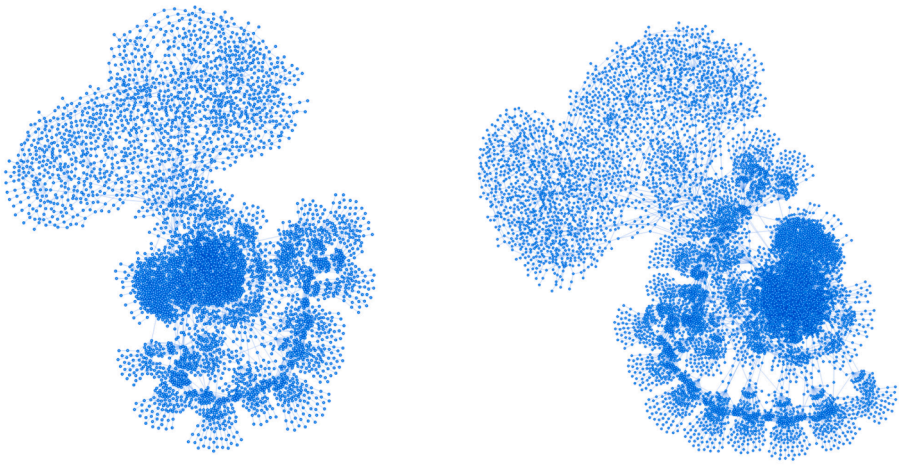
	Δ Code		Δ Objective		Δ Time		Δ Total	
Type	μ	*p*-value	μ	*p*-value	μ	*p*-value	μ	*p*-value
GP SAW	0.0105	0.8582	0.2000	0.0038*	0.1474	0.0236*	0.1193	0.0011*
GP MO	0.0211	0.7457	0.2526	0.0005*	0.1789	0.0115*	0.1509	0.0002*

Table 9
ORT, GP SAW and GP MO full search results (test set).

		ORT			GP SAW					GP MO					
Problem Class	Instance	Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ
costas-array	10	S	1		S		1				S		1			
	11	S	1		S		0				S		1			
	12	S	1		S		1				S		1			
	13	S	1		S		1				S		1			
	14	S	1		S		1				S		1			
	15	S	9		S		1	+			S		2	+		
gfd-schedule	n10f2d10m10k3	SC	1	3	SC		1		3		SC		1		3	
	n25f5d20m10k3	S	1 200	803	S		1 200		205	+	S		1 200		422	+
	n35f5d20m10k3	UNK	1 200		S	+	1 200		1 107		S	+	1 200		822	
	n55f2d50m30k3	S	1 200	2 704	S		1 200		12 507	−	S		1 200		7 155	−
	n60f7d50m30k10	S	1 200	2 215	S		1 200		19 115	−	S		1 200		9 658	−
grid-colouring	10_10	S	1 200	6	S		1 200		4	+	S		1 200		4	+
	12_13	S	1 200	7	S		1 200		6	+	S		1 200		4	+
	15_16	S	1 200	11	S		1 200		5	+	S		1 200		5	+
	5_6	SC	1	3	SC		1		3		SC		1		3	
	7_8	S	1 200	4	SC	+	1	+	3	+	SC	+	3	+	3	+
mapping	mesh2x2_1	S	1 200	1 060	SC	+	66	+	1 000	+	SC	+	5	+	1 000	+
	mesh2x2_mp3	S	1 200	1 254	SC	+	26	+	1 102	+	SC	+	175	+	1 102	+
	mesh3x3_mp3	S	1 200	1 314	S		1 200		1 436	−	S		1 200		1 262	+
	mesh4x4_1	UNK	1 200		S	+	1 200		2 564		S	+	1 200		2 354	
	ring_1	UNK	1 200		S	+	1 200		1 702		S	+	1 200		1 733	
multi-knapsack	mknap2-10	UNK	1 200		UNK		1 200				SC	+	644	+	624 319	
	mknap2-31	UNK	1 200		SC	+	25	+	9 074		SC	+	37	+	9 074	
nmseq	099	S	1		S		1				S		1			
	100	S	2		S		1				S		1			
	143	S	4		S		1	+			S		1	+		
	150	S	4		S		1	+			S		1	+		
	200	S	6		S		1	+			S		1	+		
project-planning	ProjectPlannertest_12_6	S	1 200	68	S		1 200		19	+	SC	+	2	+	17	+
	ProjectPlannertest_14_6	S	1 200	73	S		1 200		32	+	S		1 200		27	+
	ProjectPlannertest_15_8	S	1 200	66	S		1 200		42	+	SC	+	682	+	31	+
	ProjectPlannertest_16_9	S	1 200	39	S		1 200		35	+	S		1 200		35	+
	ProjectPlannertest_17_6	S	1 200	95	S		1 200		46	+	S		1 200		48	+
spot5	1502	S	1 200	64 056	S		1 200		28 043	+	S		1 200		32 050	+
	29	S	1 200	14 069	S		1 200		8 059	+	S		1 200		8 059	+
	412	S	1 200	34 457	S		1 200		34 397	+	S		1 200		39 407	−
	42	S	1 200	191 117	S		1 200		164 064	+	S		1 200		168 054	+
	5	S	1 200	331	S		1 200		275	+	S		1 200		283	+
tdtsp	inst_10_35_20	SC	638	9 055	SC		12	+	9 055		SC		9	+	9 055	
	inst_10_42_00	SC	318	8 421	SC		3	+	8 421		SC		2	+	8 421	
	inst_10_45_00	SC	1	6 819	SC		3	−	6 819		SC		2		6 819	
	inst_10_58_20	S	1 200	13 799	SC	+	11	+	10 306	+	SC	+	12	+	10 306	+
	inst_20_26_00	S	1 200	18 180	S		1 200		14 626	+	S		1 200		14 942	+
triangular	n18	S	1 200	40	S		1 200		40		S		1 200		40	
	n26	S	1 200	56	S		1 200		61	+	S		1 200		59	+
	n34	S	1 200	74	S		1 200		81	+	S		1 200		79	+
	n40	S	1 200	86	S		1 200		97	+	S		1 200		95	+
	n46	S	1 200	98	S		1 200		110	+	S		1 200		112	+
zephyrus	12_6_8_3	SC	948	1 300	S	−	1 200	−	1 300		S	−	1 200	−	3 055	−
	12_8_6_3	SC	294	1 300	S	−	1 200	−	1 300		S	−	1 200	−	6 305	−
	14_10_8_3	UNK	1 200		S	+	1 200		9 100		S	+	1 200		10 920	
	14_6_8_3	SC	247	1 170	S	−	1 200	−	8 710	−	S	−	1 200	−	5 850	−
	14_8_6_3	SC	87	1 170	S	−	1 200	−	9 230	−	S	−	1 200	−	6 760	−

Table 10
GP SAW and MO results summary by problem class (test set).

	GP SAW (Δ ORT)			GP MO (Δ ORT)		
Problem Class	−	+	Δ	−	+	Δ
costas-array	0	1	1	0	1	1
gfd-schedule	2	2	0	2	2	0
grid-colouring	0	6	6	0	6	6
mapping	1	8	7	0	9	9
multi-knapsack	0	2	2	0	4	4
nmseq	0	3	3	0	3	3
project-planning	0	5	5	0	9	9
spot5	0	5	5	1	4	3
tdtsp	1	6	5	0	6	6
triangular	0	4	4	0	4	4
zephyrus	10	1	− 9	12	1	− 11
Total	14	43	29	15	49	34



(a) ProjectPlannertest (12,7) (train) (b) ProjectPlannertest (15,6) (test)

Fig. 2. An illustration of CSP structural similarity between training and test instances.

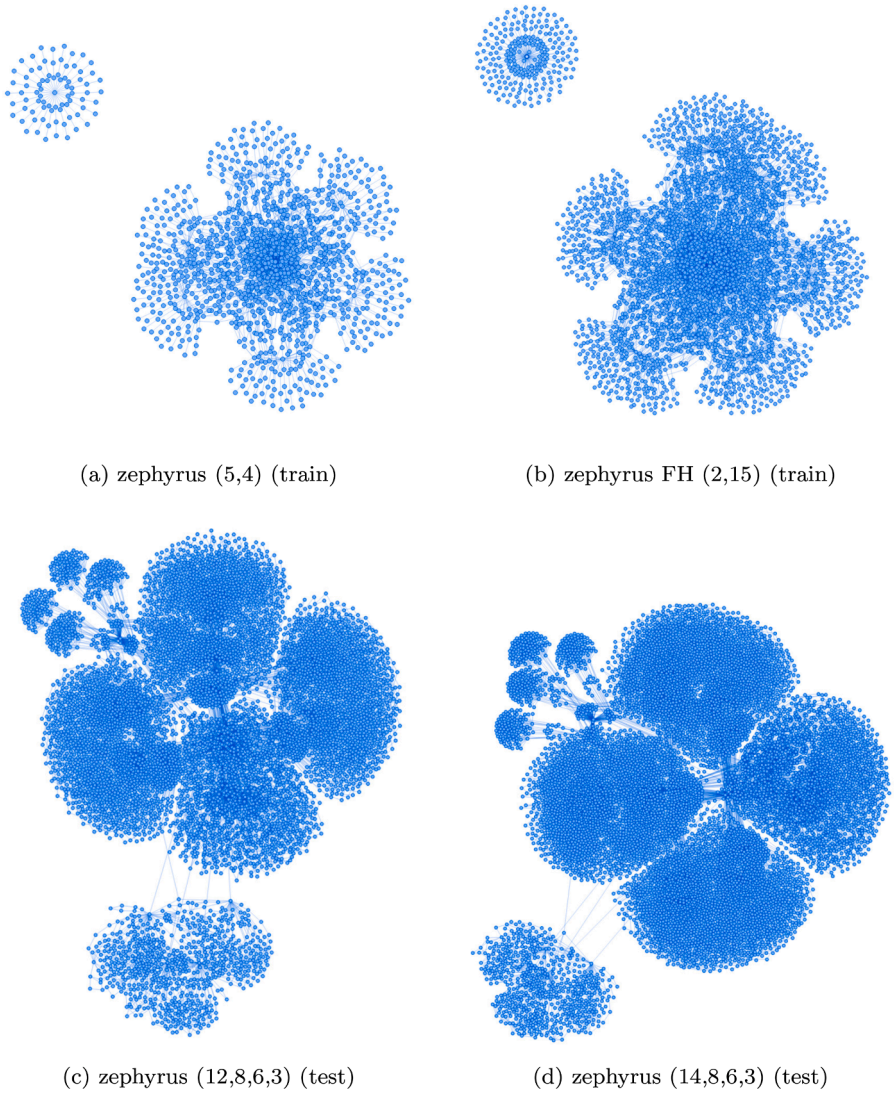
The results of the search developed on problem instances of the same class are provided in Table 9. As a general observation, the improved performance over the default ORT search on problem classes is carried forward to the test set, as summarised in Table 10. The performance of the search strategies found on the test set was largely anticipated as the methodology adopted follows that of Bennetto and Van Vuuren [3]. One problem class stands out as an exception — the zephyrus class. The zephyrus problem class warrants further discussion as strategies developed by the single-objective GP and multi-objective schemes both failed on the test set.

The CSPs in Figs. 2 and 3 are plotted according to a gravity layout model[2]. Each node in the plot represents either a variable or constraint. An example of a constraint node is simply a condition to be met between a pair of variables, such as $x_i \neq x_j$, which would result in three nodes, two of which are variable nodes (x_i, x_j) and one constraint node ($C_{\neq}$), with two edges, $(x_i, C_{\neq})$ and $(C_{\neq}, x_j)$. All nodes represent either variable-value assignments performed by the solver, or required constraint checks between such assigned values.

Two instances from the project-planning class (Fig. 2) are provided as well as two instances from the training and test data sets for the zephyrus problem class (Fig. 3) in order to illustrate the change in structure between the training and test data sets. The project-planning problem class instances shown in Fig. 2 are representative of the instances in both the training and test set. The project-planning problem class search strategies found on the training set were among the top performing strategies on the test set when compared with the default ORT search strategy.

The GP is tasked to design search strategies which preferably exploit a structural feature of the underlying graph that is expected to be present in variations of instances within the problem class. If such a structure is modified, it would not necessarily follow that the search strategy would retain its ability to perform well on a problem that no longer contains the anticipated structural exploits. Fig. 3 illustrates that the training data exhibits two connected components with a (mostly) symmetric structure emanating from a core set of variables. The graphical representation of the test set examples exhibit a single connected component with additional complexities and sub-structures within each primary region — it also appears that there are additional side constraints (clusters of constraints in the top-left portions of Figs. 3(c) and 3(d)) which were not present in the original problem definition. In addition, the parametrisation of the test data appears to have changed from a two-parameter model to a four-parameter model — which would explain the additional structural complexity observed in the test samples. This exposes a weakness in the approach employed whereby if a problem class is incorrectly classified, an unsuitable strategy may be used to solve such an instance which may perform worse than a default search with fewer structural exploitations.

[2] Using the default settings in the visNetwork R package.



(a) zephyrus (5,4) (train) (b) zephyrus FH (2,15) (train)

(c) zephyrus (12,8,6,3) (test) (d) zephyrus (14,8,6,3) (test)

Fig. 3. Structural differences between (a)–(b) training and (c)–(d) test sets for the zephyrus problem class.

5. Conclusions

The methodology presented in this paper extends the work of Minton [13], Epstein et al. [1,7] and Bennetto and Van Vuuren [3] in which a multi-objective metaheuristic approach is employed to develop branching strategies for solving classes of CSPs. To the best of the authors' knowledge, this is the first time that a multi-objective approach has been adopted to solve this particular problem.

The methodology employed reduces the specification of the strategy to an arithmetic function and adopts a classic 'Koza' style genetic programming modelling approach in conjunction with the NSGA-II to search for candidate strategies. A data set of open problem instances was employed to train the GP where problem instances were grouped by class and collectively solved. Not only does the approach demonstrate that high-quality search strategies can be found which outperform other high-ranking search strategies, including the ORT default search and single-objective GP — but that the search strategies found continue to extend to unseen problem instances of the same class.

It was found that, in one example, the structure of the test data for a particular problem class was sufficiently different to erode the performance gain observed on the training set. This finding supports the case that the GP had developed structural exploits for the class of CSPs in the training process which were rendered ineffective through a change in problem structure.

Future work may include considering a more sophisticated set of GP operators and comparing the results performance with that of the arithmetic operators in this context. A more comprehensive set of test data may also be explored.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Bain S, Thornton J, Sattar A. Methods of automatic algorithm generation. PRICAI 2004: trends in artificial intelligence, Auckland, New Zealand. 2004. p. 144–53.
[2] Bain S, Thornton J, Sattar A. Evolving variable-ordering heuristics for constrained optimisation. Principles and practice of constraint programming-CP 2005. 2005. p. 732-736.
[3] Bennetto R., Van Vuuren J.H. Evolutionary search strategies in constraint programming. Eur J Oper Res In review.
[4] Bessière C. Constraint propagation. Handb Constraint Program 2006;2:29–83.
[5] Deb K. Multi-objective optimization using evolutionary algorithmsvol. 16. New York (NY): John Wiley & Sons; 2001.
[6] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 2002;6(2):182–97.

[7] Epstein SL, Freuder EC, Wallace R, Morozov A, Samuels B. The adaptive constraint engine. International conference on principles and practice of constraint programming. Ithaca (NY): Springer; 2002. p. 525–40.
[8] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Artificial intelligence. Boston (MA): Addison-Wesley Publishing Company; 1989. ISBN 9780201157673
[9] Jefferson C., Miguel I., Hnich B., Walsh T., Gent I.P. CSPLib: a problem library for constraints. 1999. URL http://www.csplib.org.
[10] Knowles J, Corne D. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. Congress on evolutionary computation, Washington (DC). vol. 1; 1999. p. 98–105.
[11] Koza JR. Genetic programming: on the programming of computers by means of natural selectionvol. 1. Cambridge (MA): MIT Press; 1992.
[12] Luke S., Panait L., Balan G., Paus S., Skolicki Z., Bassett J., et al. ECJ: a java-based evolutionary computation research system. 2006. URL https://cs.gmu.edu/~eclab/projects/ecj/.
[13] Minton S. Automatically configuring constraint satisfaction programs: a case study. Constraints 1996;1(1-2):7–43.
[14] Perron L., Furnon V. Or-tools. 2019. URL https://developers.google.com/optimization/.
[15] Refalo P. Impact-based search strategies for constraint programming. International conference on principles and practice of constraint programming. Toronto (ON): Springer; 2004. p. 557–71.
[16] Schuurmans D, Southey F. Local search characteristics of incomplete sat procedures. Artif Intell 2001;132(2):121–50.
[17] Srinivas N, Deb K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 1994;2(3):221–48.
[18] Stewart TJ. A critical survey on the status of multiple criteria decision making theory and practice. Omega 1992;20(5-6):569–86.
[19] Stuckey PJ, Feydy T, Schutt A, Tack G, Fischer J. The minizinc challenge 2008–2013. AI Mag 2014;35(2):55–60.
[20] Wilcoxon F. Individual comparisons by ranking methods. Breakthroughs in statistics. Springer; 1992. p. 196–202.
[21] Zitzler E. Evolutionary algorithms for multiobjective optimization: methods and applicationsvol. 63. Shaker, Ithaca (NY); 1999.
[22] Zivan R, Meisels A. Conflict directed backjumping for max-csps. International joint conferences on artificial intelligence, Hyderabad. 2007. p. 198–204.