1 Solving Constrained Optimization Problems with Hybrid Evolutionary Algorithms

CARLOS COTTA AND ANTONIO J. FERNÁNDEZ
Dpto. Lenguajes y Ciencias de la Computación
University of Málaga
{ccottap,afdez}@lcc.uma.es

1.1 INTRODUCTION

The foundations for *evolutionary algorithms* (EAs) were established in the end of the 60's [1, 2] (EAs) and strengthened in the beginning of the 70's [3, 4]. EAs appeared as an alternative to the exact or approximate optimization methods whose application to many real problems were not acceptable in terms of performance. When applied to real problems, EAs provide a valuable relation between quality of the solution and efficiency to obtain it; for this reason these techniques attracted immediately the attention of many researchers and became what they nowadays represent: the cutting-edge approach to real-world optimization. Certainly, this has also been the case for other related techniques, such as *simulated annealing* [5] (SA), *tabu search* [6] (TS), etc. The term *metaheuristics* has been coined to denote them.

The term hybrid evolutionary algorithm (HEAs) (resp. hybrid metaheuristics) refers to the combination of an evolutionary technique (resp. metaheuristics) with another (perhaps exact or approximate) technique for optimization. The aim is to combine the best of both worlds with the objective of producing better results than each of the involved components working alone. HEAs have been proved to be very successful in the optimization of many practical problems (e.g., [7, 8]) and, as a consequence, currently there exist an increasing interest in the optimization community for this kind of techniques.

One crucial point in the the development of HEAs (and hybrid metaheuristics in general) is the need of exploiting problem knowledge as was clearly exposed in the formulation of the *No Free Lunch Theorem* (NFL) by Wolpert and Macready [9] (a search algorithm performs in strict accordance with the amount and quality of the problem knowledge they incorporate). Quite interestingly, this line of thinking had already been advocated by several researchers in the late 1980s and early 1990s, e.g., Hart and Belew [10], Davis [11], and Moscato [12]. Precisely this is the foundation of one of the most known instances of HEAs, the *memetic algorithms*, term firstly used in the work of Moscato [13, 14, 15] (MAs). Basically a MA is a search strategy in which a population of optimizing agents synergistically cooperate and compete [12]. These agents are explicitly concerned with using knowledge from the problem being solved, as suggested by both theory and practice [16]. The success of MA is evident and one of the consequence is that currently the term “memetic algorithm” is used as synonym of “hybrid evolutionary algorithm", although in essence MAs are a particular case of HEAs.

As already mentioned, HEAs were born to tackle many problems that are very difficult to solve using evolutionary techniques or classical approaches working alone. This is precisely the case of *constrained problems*. Generally speaking, a constrained problem consists of a set of constraints involving a number of variables restricted to have values in a set of (possibly different) finite domains; basically a constraint is a relation maintained between the entities (e.g., objects or variables) of a problem, and constraints are used to model the behavior of systems in the real world by capturing an idealized view of the interaction between the variables involved. Solving a constrained problem means the finding of a possible assignment (of values in the computation domains) for the constrained variables that satisfies all the constraints. Solving this kind of problem can be done by using different techniques ranging from traditional techniques to modern ones. For example, some approaches to solve a problem are in the area of operational research (OR), genetic algorithms, artificial intelligence (AI) techniques, rule-based computations, conventional programs and constraint-based approaches. Usually, the solving is understood as the task of searching for a single solution to the problem, although sometimes it is required to find the set of all solutions. Also, in certain cases, because of the cost of finding all solutions, the aim is just to find the best solution or an approximate solution within fixed resource bounds (e.g., in a reasonable time). Such kinds of constrained problems are called *partial constrained problems (PCPs)*. An example of a PCP is a *constrained combinatorial optimization problem (CCOP)* that assigns a cost to each solution and tries to find an optimal solution within a given time frame [17]. Precisely, this chapter focuses on CCOPs.

Not surprisingly, HEAs have been extensively used to solve this kind of problems. This chapter represents our particular share in this sense. In this work, we shall analyze the deployment of HEAs on this domain. Initially, we shall provide a generic definition for this kind of problems, and an overview of general design guidelines to tackle CCOPs. Further, we will discuss a number of interesting and not well-known[1] CCOPs that the authors have recently attacked via HEAs. The chapter will end with a summary of lessons learned, and some current and emerging research trends in HEAs for managing CCOPs.

[1]At least, in the EA community.

1.2 STRATEGIES FOR SOLVING CCOPS WITH HEAS

In general terms, an unconstrained COP is defined as a tuple $\langle S, f \rangle$, where $S \triangleq D_1 \times D_2 \times \cdots D_N$ is the search space, and $f : S \longrightarrow \mathbb{Z}$ is the objective function. Each D_i is a discrete set containing the values that a certain variable x_i can take. Therefore, any solution $\vec{x} \in S$ is a list $\langle x_1, \cdots, x_n \rangle$, and we are interested in finding the value $\vec{x}$ minimizing (wlog) the objective function f.

A constrained COP arises when we are just interested in optimizing the objective function within a subset $S_{val} \subset S$. This subset S_{val} represents the so-called *valid* or *feasible* solutions, and it may be even the case that f is a partial function defined only on these feasible solutions. The incidence vector of S_{val} on S defines a Boolean function $\phi : S \longrightarrow \mathbb{B}$, i.e., $\phi(\vec{x}) = 1$ iff $\vec{x}$ is a valid solution. In practice, constrained COPs include a well-structured function ϕ, such that it can be computed efficiently. Typically this is achieved via the conjunction of several simpler Boolean functions $\phi_i : S \longrightarrow \mathbb{B}$, i.e., $\phi(\vec{x}) = \prod_i \phi_i(\vec{x})$. Each of these functions is a *constraint.*

As an example, consider the VERTEX COVER problem: given a graph $G(V, E)$ find a subset $V' \subseteq V$ of minimal size such that for any $(u, v) \in E$ it holds that $\{u, v\} \cap V' \neq \emptyset$. In this case, $S = \{0, 1\}^{|V|}$, the objective function (to be minimized) is $f(\vec{x}) = \sum_{i=1}^{|V|} x_i$, and there is a collection of binary constraints $\phi_{uv} = \min(1, x_u + x_v)$, $(u, v) \in E$.

The previous definition can be easily generalized to weighted CCOPs, where we do not simply have to know whether a solution satisfies a particular constraint or not, but we also have an indication on *how far* that solution is from satisfying that constraint (in case it does not satisfy it at all). For this purpose, it is more convenient to denote each constraint as a function $\delta_i : S \longrightarrow \mathbb{N}$, where $\delta(\vec{x}) = 0$ indicates fulfilment of the corresponding constraint, and any strictly positive value indicates an increasingly higher degree of violation of that constraint. Note that $\phi_i(\vec{x}) = \max(0, 1 - \delta_i(\vec{x}))$.

The above formulation of weighted CCOPs allows a rather straightforward approach for tackling such a problem with HEAs, namely incorporating the constraint functions within the objective function. This can be done in different ways [18], but the simplest method is aggregating all constraint functions within a *penalty term* that is added to the objective function, e.g.,

$$f'(\vec{x}) = f(\vec{x}) + \sum_i \delta_i(\vec{x}) \tag{1.1}$$

Different variants can be defined here, such as raising each δ_i term to a certain power (thus avoiding linear compensation among constraints and/or biasing the search towards low violation degrees), or adding an offset value in case of infeasibility to ensure that any feasible solution is preferable to any non-feasible solution. This penalty approach can obviously be used only in those cases in which the objective function is defined on non-feasible solutions. A typical example is the MULTIDIMENSIONAL 0-1 KNAPSACK problem (MKP). This problem is defined via a row vector of profits $\vec{p}$, a column vector of capacity constraints $\vec{c}$, and a constraint matrix

M. Solutions are binary column vectors $\vec{x}$, the objective function is $f(\vec{x}) = \vec{p} \cdot \vec{x}$, and the feasibility constraint is $M \cdot \vec{x} \leqslant \vec{c}$. Let $\vec{d} = \vec{c} - M \cdot \vec{x}$. Then $\delta_i = -\min(0, d_i)$.

This approach has the advantage of being very simple, and allowing the use of rather standard EAs. Nevertheless, a HEA can provide notably better solution if it is aware of these penalty terms, and focuses on their optimization (see for example the MAXIMUM DENSITY STILL LIFE PROBLEM described in Section 1.3.2, or the SOCIAL GOLFER PROBLEM described in Section 1.3.3). An alternative approach is trying to enforce the search being carried within the feasible region of the search space. This can be done in two ways:

- Allowing the algorithm to temporarily traverse the infeasible region during the reproduction phase, but using a repairing procedure to turn non-feasible solutions to feasible ones before evaluation.
- Restricting the search to the feasible region at all times. This can in turn be done in two ways:
  - Defining appropriate initialization, recombination, and mutation operators that take and produce feasible solutions.
  - Defining an unconstrained auxiliary search space S_{aux} and an adequate mapping $dec : S_{aux} \longrightarrow S_{val}$.

The repair approach is possibly the simplest option after the penalty approach, although it must be noted that not always a straightforward repair procedure is available. In any case, it is interesting to note that in some sense (and depending on the particular procedure chosen for repairing), this stage can be regarded as a local search phase, and therefore some repair-based EAs can qualify as memetic algorithms. An example of repairing can be found in the GA defined by Chu and Beasley for the MKP[19]. They define a heuristic order for traversing the variables, and keep setting them to zero as long as the solution is non-feasible (this procedure is complemented by a subsequent improvement phase, in which variables are set to one in inverse order, as long as the solution remains feasible). Another example of repairing can be found in the PROTEIN STRUCTURE PREDICTION PROBLEM [20], in which the search space is composed of all embeddings of a given string in a certain fixed lattice, and solutions are only feasible if they are self-avoiding. The repairing procedure is in this case more complex, and requires the use of a backtracking algorithm to produce a feasible embedding. Nevertheless, even accounting this additional cost the HEA performs better than a simpler penalty-based approach.

Restricting the search to the feasible space via appropriate operators is in general much more complex, although it is the natural approach in certain problems, most notably in permutational problems. Feasible initialization and mutation are rather straightforward in this case, and there exists an extensive literature dealing with the definition of adequate operators for recombination, e.g., see [21]. Fixed-size subset problems are also easily dealt via this method. As to the use of decoders, it is an arguably simpler and more popular approach. A common example -again on the MKP- is to define an auxiliary search space composed of all permutations of objects, and decoding a particular permutation using a procedure that traverses the list of objects in the order indicated, and includes objects in the solution if doing so does not result in a non-feasible solution. Also in the MKP problem, a different approach was defined in [22] via the use of lists of integers representing perturbations of the original instance data, and utilizing a greedy algorithm to obtain a solution from the modified instance. Note that contrarily to some suggestions that can be found in the literature, it is not necessary (and often not even recommended) to have a decoder capable of producing any feasible solution, or producing them with the same frequency: it is perfectly admissible to ignore sub-optimal solutions, and/or biasing the search to promising regions via over-representation of certain solutions (this is precisely the case of the previous example). Another example of a general approach can be found in a GRASP-like decoding procedure [23], in which solutions are encoded via a list of natural numbers that are used to control the level of greediness o a constructive heuristic. This approach has been used with success in the GOLOMB RULER PROBLEM (see Section 1.3.1).

A particular problem of this decoder approach is the fact that locality is easily lost, that is, a small change in the genotype can result in a large change of the phenotype [24, 25]. This was observed for example in the indirect encoding of trees via Prüfer numbers [26].

1.3 STUDY CASES

This section reviews our recent work on solving CCOPs applying hybrid collaborative techniques involving evolutionary techniques. In particular we focus on four constrained problems that are not very well-known in the evolutionary programming community and that we have tackled with certain success in the recent years.

1.3.1 Optimal Golomb Rulers (OGR)

The concept of was first introduced by W.C. Babcock in 1953 [27], and further described by S.W. Golomb [28]. Golomb Rulers are a class of undirected graphs that, unlike usual rulers, measure more discrete lengths than the number of marks they carry. The particularity of Golomb Rulers that on any given ruler, all differences between pairs of marks are unique makes them really interesting in many practical applications (cf. [29, 30]).

Traditionally, researchers are interested in discovering *Optimal Golomb Ruler* (OGR), that is, the shortest Golomb ruler for a number of marks. The task of finding optimal or near-optimal Golomb rulers is computationally very hard and results in an extremely challenging combinatorial problem. Proof of it is the fact that the search for an optimal 19-marks Golomb ruler took approximately 36,200 CPU hours on a Sun Sparc workstation using a very specialized algorithm [31]. Also, optimal solutions for 20 up to 24 marks were obtained by massive parallelism projects, taking from several months up to four years (for the 24 marks instance) for each of those instances [30, 32, 33, 34].

The OGR problem can be classified as a fixed-size subset selection problem, such as e.g., the $p-$median problem [35], although exhibits some very distinctive features.

1.3.1.1 Formal definition A *n-mark Golomb ruler* is an ordered set of n distinct non-negative integers, called *marks*, $a_1 < ... < a_n$, such that all the differences $a_i - a_j$ $(i > j)$ are distinct. We have thus a number of constraints of the form $a_i - a_j \neq a_k - a_m$ $(i > j, k > m, (i,j) \neq (k,m))$. Clearly we may assume $a_1 = 0$. By convention, a_n is the *length of the Golomb ruler*. A Golomb ruler with n marks is an optimal Golomb ruler if, and only if,

- there exists no other n-mark Golomb rulers having smaller length, and
- the ruler is canonically "smaller" with respect to the the equivalent rulers. This means that the first differing entry is less than the corresponding entry in the other ruler.

Figure 1.1 shows an OGR with 4-marks. Observe that all distances between any two marks are different.

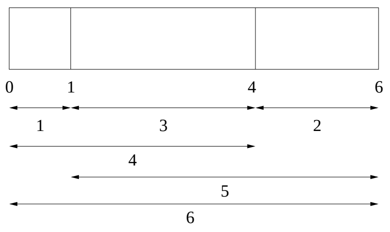

Fig. 1.1 A Golomb Ruler with 4 marks

Typically, Golomb Rulers are represented by the values of the marks on the ruler, i.e., in a n-mark Golomb ruler, $a_i = x$ $(1 \leq i \leq n)$ means that x is the mark value in position i. The sequence $(0, 1, 4, 6)$ would then represent the ruler in Figure 1.1. However, this representation turns out to be inappropriate for EAs (for example, it is problematic with respect to developing good crossover operators [36]). An alternative representation consists of representing the Golomb ruler via the lengths of its segments, where the length of a segment of a ruler is defined as the distance between two consecutive marks. Therefore, a Golomb Ruler can be represented with $n - 1$ marks specifying the lengths of the $n - 1$ segments that compose it. In the previous example, the sequence $(1, 3, 2)$ would encode the ruler depicted in Figure 1.1.

1.3.1.2 Solving OGRs In addition to already commented related work, here we discuss a variety of techniques that have been applied for finding OGRs. For instance, systematic (exact) methods, such as the method proposed by Shearer to compute OGRs up to 16 marks [37]; basically this method was based on the utilization of branch-and-bound algorithms combined with a depth first search strategy (i.e., backtracking algorithms), making use in the experiments of upper-bounds set equal to length of the best known solution. Also constraint programming (CP) techniques have also been used, although with limited success [38, 39]. The main drawback of these complete methods is that they are costly computationally (i.e., time consuming). A very interesting hybridization of local search (LS) and CP to tackle the problem was presented in [40]; up to size 13, the algorithm is run until the optimum is found, and for higher instances the quality of the solutions deteriorates.

Also (hybrid) evolutionary algorithms were applied to this problem In this case, two main approaches can be essentially considered for tackling the OGR problem with EAs. The first one is the *direct* approach, in which the EA conducts the search in the space $\mathcal{S}_G$ of all possible Golomb rulers. The second one is the *indirect* approach, in which an auxiliary $\mathcal{S}_{aux}$ space is used by the EA. In this latter case, a decoder [41] must be utilized in order to perform the $\mathcal{S}_{aux} \longrightarrow \mathcal{S}_G$ mapping. Examples of the former (direct) approach are the works of Soliday *et al.* [36], and Feeney [29]. As to the latter (indirect) approach, we can cite the work by Pereira *et al.* [42] (based on the notion of random-keys [43]); also Cotta and Fernández [23] used a problem-aware procedure (inspired in GRASP [44]) to perform the genotype-to-phenotype mapping wiht the aim of ensuring the generation of feasible solutions; this method was shown to outperform other previous approaches.

A HEA incorporating a tabu search algorithm for mutation was proposed in [45]. The basic idea was to optimize the length of the rulers indirectly by solving a sequence of feasibility problems (starting from an upper bound l and producing a sequence of rulers of length $l_1 > l_2 > \ldots > l_i > \ldots$). This algorithm performed very efficiently and was able to find OGRs for up to 14 marks; in any case we make note that this method requires an estimated initial upper bound, something that clearly favored its efficiency. At the same time, we conducted a theoretical analysis on the problem trying to shed some light on the question of what makes a problem hard for a certain search algorithm for the OGR problem. This study, published in [46], consisted of an analysis of the fitness landscape of the problem. Our analysis indicated that the high irregularity of the neighborhood structure for the direct formulation introduces a drift force towards low-fitness regions of the search space. The indirect formulation that we had previously considered in [23] does not have this drawback, and hence would be in principle more amenable for conducting local search in it. Then in [47] we presented a MA in which our indirect approach (i.e., that GRASP-based EA) was used in the phases of initialization and restarting of the population whereas a direct approach, in form of a local improvement method based on the tabu search (TS) algorithm described in [45], was considered in the stages of recombination and local improvement; Experimental results showed that this algorithm could solve OGRs up to 15 marks, and produced Golomb rulers for 16 marks that are very close to the optimal value (i.e., 1.1% far), thus significantly improving the results reported in the EA literature.

Recently, in [48] we have combined ideas from greedy randomized adaptive search procedures (GRASP) [49], scatter search (SS) [50, 51], tabu search (TS) [52, 53], clustering [54], and constraint programming (CP), and the resulting algorithm was able of solving the OGR problem for up to 16 marks, a notorious improvement with regard to previous approaches reported in the literature in all those mentioned areas (i.e., EAs, LS, and CP). This algorithm yields a metaheuristic approach which is currently a state-of-the-art method compared to other metaheuristic approaches.

1.3.2 The Maximum Density Still Life Problem (MDSLP)

Conway's game of life [55] consists of an infinite checkerboard in which the only player places checkers on some of its squares. Each square has eight neighbors: the eight cells that share one or two corners with it. A cell is alive if there is a checker on it, and dead otherwise. The state of the board evolves iteratively according to three rules: (i) if a cell has exactly two living neighbors then its state remains the same in the next iteration, (ii) if a cell has exactly three living neighbors then it is alive in the next iteration, and (iii) if a cell has fewer than two or more than three living neighbors, then it is dead in the next iteration.

One challenging constraint optimization problem based on the game of life is the *maximum density still life problem* (MDSLP). In order to introduce this problem, let us define a stable pattern (also called a *still life*) as a board configuration that does not change through time, and let the *density* of a region be its percentage of living cells. The MDSLP in an $n \times n$ grid consists of finding a *still life* of maximum density. This problem is very hard to solve, and though it has not been proven to be NP-hard to the best of our knowledge, no polynomial-time algorithm for it is known. The problem has a number of interesting applications [56, 57, 58], and a dedicated web page[2] maintains up-to-date results. Figure 1.2 shows some maximum density still lifs for small values of n.

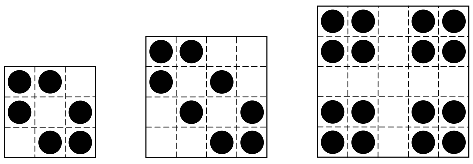

Fig. 1.2 Maximum density still lifes for $n \in \{3, 4, 5\}$.

1.3.2.1 Formal definition The constraints and objectives of the MDSLP are formalized in this section in which we follow a similar notation to the one used in [59, 60]. To state the problem formally, let r be an $n \times n$ binary matrix, such that $r_{ij} \in \{0, 1\}, 1 \leq i, j \leq n$ ($r_{ij} = 0$ if cell (i, j) is dead, and 1 otherwise). In addition, let $\mathcal{N}(r, i, j)$ be the set comprising the neighborhood of cell r_{ij}:

[2]http://www.ai.sri.com/ nysmith/life

$$\mathcal{N}(r,i,j) = \{\, r_{(i+x)(j+y)} \,|\, x,y \in \{-1,0,1\} \wedge x^2+y^2 \neq 0 \\ \wedge\, 1 \leq (i+x),(j+y) \leq \|r\| \,\} \tag{1.2}$$

where $\|r\|$ denotes the number of rows (or columns) of square matrix r, and let the number of living neighbors for cell r_{ij} be noted as $\eta(r,i,j)$:

$$\eta(r,i,j) = \sum_{c \in \mathcal{N}(r,i,j)} c \tag{1.3}$$

According to the rules of the game, let us also define the following predicate that checks whether cell r_{ij} is stable:

$$S(r,i,j) = \begin{cases} 2 \leq \eta(r,i,j) \leq 3, & r_{ij} = 1 \\ \eta(r,i,j) \neq 3, & r_{ij} = 0. \end{cases} \tag{1.4}$$

In order to check boundary conditions, we will further denote by $\widetilde{r}$ the $(n+2) \times (n+2)$ matrix obtained by embedding r in a frame of dead cells:

$$\widetilde{r}_{ij} = \begin{cases} r_{(i-1)(j-1)}, & 2 \leq i,j \leq n+1 \\ 0, & \text{otherwise.} \end{cases} \tag{1.5}$$

The maximum density still life problem for an $n \times n$ board, MDSLP(n), can now be stated as finding an $n \times n$ binary matrix r, such that

$$\sum_{1 \leq i,j \leq n} (1 - r_{ij}) \text{ is minimal,} \tag{1.6}$$

subject to

$$\bigwedge_{1 \leq i,j \leq n+2} S(\widetilde{r},i,j). \tag{1.7}$$

1.3.2.2 Solving MDSLPs The MDSLP has been tackled using different approaches. Bosch and Trick [61] compared different formulations for the MDSLP using integer programming (IP) and constraint programming (CP). Their best results were obtained with a hybrid algorithm mixing the two approaches. They were able to solve the cases for $n = 14$ and $n = 15$ in about 6 and 8 days of CPU time respectively. Smith [62] used a pure constraint programming approach to undertake the problem and proposed a formulation of the problem as a constraint satisfaction problem with 0-1 variables and non-binary constraints; only instances up to $n = 10$ could be solved. The best results for this problem were reported by Larrosa *et al.* in [59, 60], that showed the usefulness of an exact technique based on variable elimination and commonly used for solving constraint satisfaction problems: the *bucket elimination* (BE) [63]. Their basic approach could solve the problem for $n = 14$ in about 10^5 seconds. Further improvements pushed the solvability boundary forward to $n = 20$ in about twice as much time. Recently, Cheng and Yap [64, 65] have tackled the problem via the use of ad-hoc global `case` constraints, but their results are are far from the ones obtained previously by Larrosa *et al.*

Note that all the previously discussed techniques are exact approaches inherently limited for increasing problem sizes, and whose capabilities as anytime algorithms are unclear. To avoid this limitation, we recently proposed the use of HEAs to tackle this problem. Particularly, in [66] we considered a MA consisting of an EA endowed with tabu search, and where BE is used as a mechanism for recombining solutions, providing the best possible child from the parental set. Experimental tests indicated that the algorithm provided optimal or near-optimal results at an acceptable computational cost. A subsequent paper [67] dealt with expanded multi-level models in which our previous exact/metaheuristic hybrid was further hybridized with a branch-and-bound derivative, namely Beam Search (BS). The experimental results shown that our hybrid evolutionary proposals were a practical and an efficient alternative to the exact techniques employed so far to obtain still life patterns

Recently, in [48] we proposed a new hybrid algorithm that uses the technique of Mini-Buckets (MB) [68] to further improve the lower bounds of the partial solutions that are considered in the BS part of the hybrid algorithm. This new algorithm is obtained from the hybridization, at different levels, of complete solving techniques (BE), incomplete deterministic methods (BS and MB) and stochastic algorithms (MAs). An experimental analysis showed that this new proposal consistently finds optimal solutions for MDSLP instances up to $n = 20$ in considerably less time than all the previous approaches reported in the literature. Moreover, this HEA performed at the state-of-the-art, providing solutions that are equal or better to the best ones reported to date in the literature.

1.3.3 The Social Golfer Problem (SGP)

The social golfer problem (SGP) was first posted on `sci.op-research` in May 1998, and it consists of scheduling $n = g \cdot s$ golfers into g groups of s players every week for w weeks so that no two golfers play in the same group more than once. The problem can be regarded as an optimization problem if for two given values for g and s, we ask for the maximum number of weeks w the golfers can play together. SGP is a combinatorial constrained problem that raises interesting issues in symmetry breaking (e.g., players can be permuted within groups, groups can be ordered arbitrarily within every week, and even the weeks themselves can be permuted). Note that symmetry is also present in both the OGR problem and the MDSLP. Notice however that problem symmetry is beyond the scope of this paper and thus symmetry issues will not be explicitly discussed here.

1.3.3.1 Formal definition As mentioned above, the Social Golfer Problem (SGP) consists of scheduling $n = g \cdot s$ golfers into g groups of s players every week for w weeks, so that no two golfers play in the same group more than once. An instance of the social golfer is thus specified by a triplet $\langle g, s, w\rangle$. A (potentially infeasible) solution for such an instance is given by a schedule $\sigma : \mathbb{N}_g \times \mathbb{N}_w \longrightarrow 2^{\mathbb{N}_n}$, where $\mathbb{N}_i = \{1, 2, \cdots, i\}$, and $|\sigma(i,j)| = s$ for all $i \in \mathbb{N}_g$, $j \in \mathbb{N}_w$, that is, a function that on input (i,j) returns the set of s players that constitute the i-th group of the j-th week. There are many possible modelings for the social golfer problem, which is one of the reasons why it is so interesting. In a generalized way, this problem can be modelled as a constraint satisfaction problem (CSP) defined by the following constraints:

- A golfer plays exactly once a week, i.e.,

$$\forall p \in \mathbb{N}_n : \forall j \in \mathbb{N}_w : \exists! i \in \mathbb{N}_g : p \in \sigma(i,j). \tag{1.8}$$

  This constraint can be also formalized by claiming that no two groups in the same week intersect, i.e.,

$$\forall j \in \mathbb{N}_w : \forall i, i' \in \mathbb{N}_g, i \neq i' : \sigma(i,j) \cap \sigma(i',j) = \emptyset. \tag{1.9}$$

- No two golfers play together more than once, i.e.,

$$\forall j, j' \in \mathbb{N}_w : \forall i, i' \in \mathbb{N}_g, i \neq i' : |\sigma(i,j) \cap \sigma(i',j')| \leqslant 1. \tag{1.10}$$

  This constraint can also be formulated as a weighted constraint: let $\#_\sigma(a,b)$ be the number of times golfers a and b play together in schedule σ, i.e.,

$$\#_\sigma(a,b) = \sum_{i \in \mathbb{N}_g} \sum_{j \in \mathbb{N}_w} \left[\{a,b\} \subseteq \sigma(i,j) \right], \tag{1.11}$$

  where $[\cdot]$ is the Iverson bracket, namely [`true`]$= 1$ and [`false`]$= 0$. Then, we can define the degree of violation of a constraint a-and-b-play-together-at-most-once as $\max(0, \#_\sigma(a,b) - 1)$.

As already commented symmetries can appear in (and can be removed from) this problem in several forms; see [69, 70, 71] for more details.

1.3.3.2 Solving the SGP The SGP was firstly attacked by CP techniques that mainly addressed the SGP by detecting and breaking symmetries (e.g., [72, 73, 74, 75, 76, 77, 78, 70]), Due to the interesting properties of the problem, it also attracted the attention in another optimization areas, and has been extensively tackled via different techniques. Here, we mention just some of the most recent advances in solving the SGP. For instance, Harvey and Winterer [69] have proposed to construct solutions to the SGP by using sets of mutually orthogonal latin squares. Also, Gent and Lynce [79] have recently introduced a satisfiability (SAT) encoding for the SGP. Barnier and Brisset [70] have presented a combination of techniques to efficiently find solutions to a specific instance of SGP, the Kirkman's schoolgirl problem. Global constraints for lexicographic ordering have been proposed by Frisch *et al.* [80], being used for breaking symmetries in the SGP. Also, a tabu-based local search algorithm for the SGP is described by Dotú and Van Hentenryck [81].

In [71], we presented, to the best of our knowledge, the first attempt of tackling the SGP by evolutionary techniques; it consisted of a memetic algorithm (MA) that is based on the hybridization of evolutionary programming and tabu search. The flexibility of MAs eased the handling of the problem symmetries. Our MA, based on selection, mutation and local search performed at a state-of-the-art level for this problem.

1.3.4 The Consensus Tree Problem (CTP)

The inference (or reconstruction) of phylogenetic trees is a problem from the bioinformatics domain that has direct implications in areas such as multiple sequence alignment[82], protein structure prediction[83] or molecular epidemiological studies of viruses[84], just to cite a few. This (optimization) problem seeks the best tree representing the evolutionary history of a collection of species, providing therefore, a hierarchical representation of the degree of closeness among a set of organisms. This is typically done on the basis of molecular information –e.g., DNA sequences– from these species, and can be approached in a number of ways: maximum likelihood, parsimony, distance matrices, etc. [85]. A number of different high-quality trees (with quality measured in different ways) can then be found, each possibly telling something about the *true* solution. Furthermore, the fact that data come from biological experiments, which are not exact, makes near-optimal solutions (even near-optimal with respect to different criteria) be almost as relevant as the actual optimum. It is in this situation where the *consensus tree* (often called *supertree*) problem comes into play [86]. Essentially, a consensus method tries to summarize a collection of trees provided as input, returning a single tree [87]. This implies identifying common substructures in the input trees and representing these in the output tree.

1.3.4.1 Formal definition Let T be a strictly binary rooted tree; a LISP-like notation will be used to denote the structure of the tree. Thus, (sLR) is the tree with root s, and with L and R as subtrees, and () is an empty tree. The notation (a) is a shortcut for $(a()())$. Let $\mathcal{L}(T)$ be the set of leaves of T. Each edge e in T defines a bipartition $\pi_T(e) = \langle S_1, S_2 \rangle$, where S_1 are the leaves in $\mathcal{L}(T)$ that can be reached from the root passing through e, and S_2 are the remaining leaves. We define $\Pi(T) = \{\pi_T(e) \mid e \in T\}$.

The consensus tree problem consists of representing a collection of trees $\{T_1, \cdots, T_m\}$ as a single tree that should be optima with respect to certain model. This model can be approached in several ways [87]; for instance, the models described in [88] (i.e., tree compatibility problem) and in [89] (i.e., strict consensus) focus on finding a tree such that $\Pi(T) = \cup_{i=1}^{m} \Pi(T_i)$ and $\Pi(T) = \cap_{i=1}^{m} \Pi(T_i)$ respectively; also the model presented in [90] (i.e.,the median tree) tries to minimize the sum of differences between T and the input trees, i.e., $\min \sum_{i=1}^{m} d(T, T_i)$.

As a consequence, the meaning of global optimum is different from typical CCOPs since it is very dependant on the selected model as well as another metrics such as the way of evaluating the differences between the trees (i.e., the distance between trees). This is so because the distance $d(T, T')$ between trees is not standard, and different alternatives can be considered. Perhaps the most typical distance is defined as the number of non-common bipartitions in $\Pi(T)$ and $\Pi(T')$: this is also termed the partition metric. In any case, alternative metrics can be considered. For example, one can cite edit distance (measured in terms of some edition operations, such as nearest neighbor interchange (NNI), subtree prune and regraft (SPR), or tree bisection and reconnection (TBR) among others – see [91]), or the TreeRank measure [92]. All of these metrics have advantages and drawbacks so that is not always easy to determine which is the best election.

The fact that the problem admits different formulations is an additional reason making it so interesting. For instance, recently, the problem was formulated as a constraint satisfaction problem in [93]: Gent *et al.* presented a constraint encoding based on the observation that any rooted tree can be considered as being min-ultrametric [94] when we label interior nodes with their depth in that tree. This guarantees that any path from the root to a leaf corresponds to a strictly increasing sequence. See [93] and [95] for more details about this encoding.

1.3.4.2 Solving the CTP Regarding the inference of phylogenetic trees, the use of classical exact techniques can be considered generally inappropriate in this context. Indeed, the use of heuristic techniques in this domain seems much more adequate. These can range from simple constructive heuristics (e.g., greedy agglomerative techniques such as UPGMA[96]) to complex metaheuristics [97] (e.g., evolutionary algorithms[98, 99] or local search [100]). At any rate, it is well-known that any heuristic method is going to perform in strict accordance with the amount of problem-knowledge it incorporates[16, 9]. In [101] we precisely explored this possibility, and presented a model for the integration of branch-and bound techniques (BnB)[102] and memetic algorithms (MAs)[13, 14, 15]. This model resulted in a synergistic combination yielding better results (experimentally speaking) than each of its constituent techniques, as well as classical agglomerative algorithms and other efficient tree-construction methods.

Regarding the consensus tree problem, very different methods have been applied (e.g., polynomial time algorithms [103], constraint programming (CP) [93, 95, 104], and evolutionary algorithms (EA) [99] among others). With the aim of improving the so-far-obtained results we are now experimenting with hybrid methods that combine the best of the CP and EA proposals. More specifically, we are evaluating two algorithms: the first one is an MA [15] in which we use a CP method for supertree construction based in the method published in [93] as recombination operator. The results should be reported soon [105].

1.4 CONCLUSIONS

Constrained COPs are ubiquitous, and are representative of a plethora of relevant real-world problems. As such, they are also typically hard to solve, and demand the use of flexible cutting-edge optimization technologies for achieving competitive results. As we have shown in this work, the framework of evolutionary computation offers a solid ground upon which powerful optimizations algorithms for CCOPs can be built via hybridization.

HEAs provide different options for dealing with CCOPs, ranging from the inclusion of constraints as side objectives to the definition of *ad hoc* operators working within the feasible region, including the use of repairing mechanisms or complex genotype-phenotype mappings. Each of these approaches is suited to different –not necessarily disjoint– optimization scenarios. Hence, they can avail the practitioner in different ways, providing her with alternative methods for solving the problem at hand.

It is difficult to provide general design guidelines, since there are many problem specificities to be accounted. The availability of other heuristics (either classical or not) may suggest the usefulness of performing a direct search in the space of solutions, so that these heuristics can be exploited by the HEA. The particulars of the objective function or the sparseness of valid solutions may dictate whether a penalty-based approach is feasible or not (e.g., non-feasible solutions could not be evaluated at all, the search algorithm could spend most of the time outside the feasible region, etc.). The availability of construction heuristics can in turn suggest the utilization of indirect approaches, e.g., via decoder functions. This approach is sometimes overestimated, in the sense that one has to be careful to provide the HEA with a search landscape which is easy to navigate. Otherwise, the benefits of the heuristic mapping can be counteracted by the erratic search dynamics.

From a global perspective, the record of success of HEAs on CCOPs suggests that these techniques will be increasingly used to solve problems in this domain. It is then expected that the corpus of theoretical knowledge on the deployment of HEAs on CCOPs will grow alongside with the applications of these techniques to new problems in the area.

Acknowledgments

The authors are partially supported by the Ministry of Science and Technology and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project). Second author is also supported under contract TIN-2007-67134.

References

1. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

2. H.-P. Schwefel. *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*. Diplomarbeit, Technische Universität Berlin, Hermann Föttinger–Institut für Strömungstechnik, März 1965.

3. J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

4. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

5. S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

6. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.

7. C. Reeves. Hybrid genetic algorithm for bin-packing and related problems. *Annals of Operation Research*, 63:371–396, 1996.

8. A.P. French, A.C. Robinson, and J.M. Wilson. Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7(6):551–564, 2001.

9. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

10. W. E. Hart and R. K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, pages 190–195, San Mateo CA, 1991. Morgan Kaufmann.

11. L. D. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Computer Library, New York, 1991.

12. P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent

Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

13. P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill, Maidenhead, Berkshire, England, UK, 1999.

14. P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, Boston MA, 2003.

15. P. Moscato, C. Cotta, and A. S. Mendes. Memetic algorithms. In G. C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*, pages 53–85. Springer-Verlag, Berlin Heidelberg, 2004.

16. J. Culberson. On the futility of blind search: An algorithmic view of “No Free Lunch”. *Evolutionary Computation*, 6(2):109–127, 1998.

17. E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(21-70):21–70, 1992.

18. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computating*. Natural Computing Series. Springer, 2003.

19. P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

20. C. Cotta. Protein structure prediction using evolutionary algorithms hybridized with backtracking. In J. Mira and J.R. Álvarez, editors, *Artificial Neural Nets Problem Solving Methods*, volume 2687 of *Lecture Notes in Computer Science*, pages 321–328, Berlin Heidelberg, 2003. Springer-Verlag.

21. C. Cotta and J.M. Troya. Genetic forma recombination in permutation flowshop problems. *Evolutionary Computation*, 6(1):25–44, 1998.

22. C. Cotta and J.M. Troya. A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 3*, pages 251–255, Wien New York, 1998. Springer.

23. C. Cotta and A.J. Fernández. A hybrid GRASP - evolutionary algorithm approach to Golomb ruler search. In Xin Yao et al., editors, *Parallel Problem Solving From Nature VIII*, number 3242 in Lecture Notes in Computer Science, pages 481–490, Berlin Heidelberg, 2004. Springer.

24. E. Falkenauer. The lavish ordering genetic algorithm. In *Proceedings of the 2nd Metaheuristics International Conference (MIC-97)*, Sophia-Antipolis, France, 1997.

25. J. Gottlieb and G.R. Raidl. The effects of locality on the dynamics of decoder-based evolutionary search. In L.D. Whitley et al., editors, *Proceedings of the 2000*

Genetic and Evolutionary Computation Conference, pages 283–290, Las Vegas, Nevada, USA, 2000. Morgan Kaufmann.

26. J. Gottlieb, B.A. Julstrom, F. Rothlauf, and G. R. Raidl. Prüfer numbers: A poor representation of spanning trees for evolutionary search. In L. Spector et al., editors, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 343–350. Morgan Kaufmann, 2001.

27. W.C. Babcock. Intermodulation interference in radio systems. *Bell Systems Technical Journal*, pages 63–73, 1953.

28. G.S. Bloom and S.W. Golomb. Aplications of numbered undirected graphs. *Proceedings of the IEEE*, 65(4):562–570, 1977.

29. B. Feeney. Determining optimum and near-optimum Golomb rulers using genetic algorithms. Master thesis, Computer Science, University College Cork, October 2003.

30. W.T. Rankin. Optimal Golomb rulers: An exhaustive parallel search implementation. Master thesis, Duke University Electrical Engineering Dept., Durham, NC, December 1993.

31. A. Dollas, W. T. Rankin, and D. McCracken. A new algorithm for Golomb ruler derivation and proof of the 19 mark ruler. *IEEE Transactions on Information Theory*, 44:379–382, 1998.

32. M. Garry, D. Vanderschel, et al. In search of the optimal 20, 21 & 22 mark Golomb rulers. GVANT project, `http://members.aol.com/golomb20/index.html`, 1999.

33. J. B. Shearer. Golomb ruler table. Mathematics Department, IBM Research, `http://www.research.ibm.com/people/s/shearer/grtab.html`, 2001.

34. W. Schneider. Golomb rulers. MATHEWS: The Archive of Recreational Mathematics, `http://www.wschnei.de/number-theory/golomb-rulers.html`, 2002.

35. P. Mirchandani and R. Francis. *Discrete Location Theory*. Wiley-Interscience, 1990.

36. S.W. Soliday, A. Homaifar, and G.L. Lebby. Genetic algorithm approach to the search for Golomb rulers. In L.J. Eshelman, editor, *6th International Conference on Genetic Algorithms (ICGA'95)*, pages 528–535, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.

37. J.B. Shearer. Some new optimum Golomb rulers. *IEEE Transactions on Information Theory*, 36:183–184, January 1990.

38. B.M. Smith and T. Walsh. Modelling the Golomb ruler problem. In *Workshop on non-binary constraints (IJCAI'99)*, Stockholm, 1999.

39. P. Galinier, B. Jaumard, R. Morales, and G. Pesant. A constraint-based approach to the Golomb ruler problem. In *3rd International Workshop on Integration of AI and OR Techniques (CP-AI-OR'2001)*, 2001.

40. S. Prestwich. Trading completeness for scalability: Hybrid search for cliques and rulers. In *Third International Workshop on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-01)*, pages 159–174, Ashford, Kent, England, 2001.

41. S Koziel and Z. Michalewicz. A decoder-based evolutionary algorithm for constrained parameter optimization problems. In T. Bäeck, A.E. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 231–240. Springer, Berlin Heidelberg, 1998.

42. F.B. Pereira, J. Tavares, and E. Costa. Golomb rulers: The advantage of evolution. In F. Moura-Pires and S. Abreu, editors, *Progress in Artificial Intelligence, 11th Portuguese Conference on Artificial Intelligence*, number 2902 in Lecture Notes in Computer Science, pages 29–42, Berlin Heidelberg, 2003. Springer.

43. J. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.

44. M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, Boston MA, 2003.

45. I. Dotú and P. Van Hentenryck. A simple hybrid evolutionary algorithm for finding golomb rulers. In D.W. Corne et al., editors, *2005 Congress on Evolutionary Computation (CEC2005)*, volume 3, pages 2018–2023, Edinburgh, Scotland, 2005. IEEE.

46. C. Cotta and A.J. Fernández. Analyzing fitness landscapes for the optimal golomb ruler problem. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3248 of *Lecture Notes in Computer Science*, pages 68–79, Berlin Heidelberg, 2005. Springer.

47. C. Cotta, I. Dotú, A.J. Fernández, and P. Van Hentenryck. A memetic approach to golomb rulers. In T.P. Runarsson et al., editors, *Parallel Problem Solving From Nature IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 252–261, Berlin Heidelberg, 2006. Springer-Verlag.

48. C. Cotta, I. Dotú, A.J. Fernández, and P. Van Hentenryck. Local search-based hybrid algorithms for finding golomb rulers. *Constraints*, 12(3):263–291, 2007.

49. T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

50. F. Glover. A template for scatter search and path relinking. *Lecture Notes in Computer Science*, 1363:13–54, 1997.

51. M. Laguna and R. Martí. *Scatter Search. Methodology and Implementations in C.* Kluwer Academic Publishers, Boston MA, 2003.

52. F. Glover. Tabu search – part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.

53. F. Glover. Tabu search – part II. *ORSA Journal of Computing*, 2(1):4–31, 1989.

54. A.K. Jain, N.M. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

55. M. Gardner. The fantastic combinations of John Conway's new solitaire game. *Scientific American*, 223:120–123, 1970.

56. M. Gardner. On cellular automata, self-reproduction, the garden of Eden and the game of "life". *Scientific American*, 224:112–117, 1971.

57. E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*, volume 2 of *Games in Particular*. Academic Press, London, 1982.

58. M. Gardner. *Wheels, Life, and Other Mathematical Amusements*. W.H. Freeman, New York, 1983.

59. J. Larrosa and E. Morancho. Solving 'still life' with soft constraints and bucket elimination. In Rossi [106], pages 466–479.

60. J. Larrosa, E. Morancho, and D. Niso. On the practical use of variable elimination in constraint optimization problems: 'still life' as a case study. *Journal of Artificial Intelligence Research*, 23:421–440, 2005.

61. R. Bosch and M. Trick. Constraint programming and hybrid formulations for three life designs. In *CP-AI-OR*, pages 77–91, 2002.

62. B. M. Smith. A dual graph translation of a problem in 'life'. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP'2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 402–414, Ithaca, NY, USA, 2002. Springer.

63. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

64. K. C. K. Cheng and R. H. C. Yap. Ad-hoc global constraints for life. In Peter van Beek, editor, *Principles and Practice of Constraint Programming – CP'2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 182–195, Sitges, Spain, 2005. Springer.

65. K. C. K. Cheng and R. H. C. Yap. Applying ad-hoc global constraints with the case constraint to still-life. *Constraints*, 11:91–114, 2006.

66. J.E. Gallardo, C. Cotta, and A.J. Fernández. A memetic algorithm with bucket elimination for the still life problem. In J. Gottlieb and G. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3906 of *Lecture Notes in Computer Science*, pages 73–85, Berlin Heidelberg, 2006. Springer-Verlag.

67. J.E. Gallardo, C. Cotta, and A.J. Fernández. A multi-level memetic/exact hybrid algorithm for the still life problem. In T.P. Runarsson et al., editors, *Parallel Problem Solving from Nature IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 212–221, Berlin Heidelberg, 2006. Springer-Verlag.

68. R. Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *IJCAI*, pages 1297–1303, 1997.

69. W. Harvey and T. Winterer. Solving the MOLR and social golfers problems. In Peter van Beek, editor, *11th International Conference on Principles and Practice of Constraint Programming*, volume 3709 of *Lecture Notes in Computer Science*, pages 286–300, Sitges, Spain, 2005. Springer.

70. N. Barnier and P. Brisset. Solving kirkman's schoolgirl problem in a few seconds. *Constraints*, 10(1):7–21, 2005.

71. C. Cotta, I. Dotú, A.J. Fernández, and P. Van Hentenryck. Scheduling social golfers with memetic evolutionary programming. In Francisco Almeida et al., editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 150–161, Berlin Heidelberg, 2006. Springer-Verlag.

72. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In Toby Walsh, editor, *7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 93–107, Paphos, Cyprus, 2001. Springer.

73. B. M. Smith. Reducing symmetry in a combinatorial design problem. In *Third International Workshop on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 351–359, 2001.

74. M. Sellmann and W. Harvey. Heuristic constraint propagation. In Pascal Van Hentenryck, editor, *8th International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 738–743, Ithaca, NY, USA, 2002. Springer.

75. A. Ramani and I.L. Markov. Automatically exploiting symmetries in constraint programming. In B. Faltings, A. Petcu, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004*, volume 3419 of *Lecture Notes in Computer Science*, pages 98–112, Lausanne, Switzerland, 2005. Springer. Revised Selected and Invited Papers.

76. S.D. Prestwich and A. Roli. Symmetry breaking and local search spaces. In R. Barták and M. Milano, editors, *Second International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3524 of *Lecture Notes in Computer Science*, pages 273–287, Prague, Czech Republic, 2005. Springer.

77. T. Mancini and M. Cadoli. Detecting and breaking symmetries by reasoning on problem specifications. In J.-D. Zucker and L. Saitta, editors, *International Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, volume 3607 of *Lecture Notes in Computer Science*, pages 165–181, Airth Castle, Scotland, UK, 2005. Springer.

78. M. Sellmann and P. Van Hentenryck. Structural symmetry breaking. In L. Pack Kaelbling and A. Saffiotti, editors, *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 298–303, Edinburgh, Scotland, 2005. Professional Book Center.

79. I.P. Gent and I. Lynce. A SAT encoding for the social golfer problem. In *IJCAI'05 workshop on Modelling and Solving Problems with Constraints*, Edinburgh, Scotland, July 2005.

80. A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In P. Van Hentenryck, editor, *8th International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 93–108, Ithaca, NY, USA, 2002. Springer.

81. I. Dotú and P. Van Hentenryck. Scheduling social golfers locally. In R. Barták and M. Milano, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems 2005*, volume 3524 of *Lecture Notes in Computer Science*, pages 155–167, Prague, Czech Republic, 2005. Springer-Verlag.

82. J. Hein. A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular Biology and Evolution*, 6:649–668, 1989.

83. B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232:584–599, 1993.

84. C.-K. Ong, S. Nee, A. Rambaut, H.-U. Bernard, and P.H. Harvey. Elucidating the population histories and transmission dynamics of papillomaviruses using phylogenetic trees. *Journal of Molecular Evolution*, 44:199–206, 1997.

85. J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. In T. Lengauer et al., editors, *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, Heidelberg, 1999. The American Association for Artificial Intelligence Press.

86. O. R. P. Bininda-Emonds, editor. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*. Computational Biology Series. Kluwer, 2004.

87. D. Bryant. A classification of consensus methods for phylogenetics. In M. Janowitz et al., editors, *Bioconsensus*, pages 163–184. DIMACS-AMS, 2003.

88. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

89. W.H.E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classiffication*, 2:7–28, 1985.

90. J.-P. Barthélemy and F.R. McMorris. The median procedure for n-trees. *Journal of Classiffication*, 3:329–334, 1986.

91. B.L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5:1–15, 2001.

92. J.T.L. Wang, H. Shan, D. Shasha, and W.H. Piel. Treerank: A similarity measure for nearest neighbor searching in phylogenetic databases. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 171–180, Cambridge MA, 2003. IEEE Press.

93. I.P. Gent, P. Prosser, B.M. Smith, and W. Wei. Supertree construction with constraint programming. In Rossi [106], pages 837–841.

94. B.Y. Wu, K.-M. Chao, and C.Y. Tang. Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. *Journal of Combinatorial Optimization*, 3(2):199–211, 1999.

95. P. Prosser. Supertree construction with constraint programming: recent progress and new challenges. In *WCB06 - Workshop on Constraint Based Methods for Bioinformatics*, pages 75–82. N/A, 2006.

96. R.R. Sokal and C.D. Michener. A statistical method for evaluating systematic relationships. *University Kansas Science Bulletin*, 38:1409–1438, 1958.

97. A.A. Andreatta and C.C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8:429–447, 2002.

98. C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. In J.J. Merelo et al., editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 720–729. Springer-Verlag, Berlin, 2002.

99. C. Cotta. On the application of evolutionary algorithms to the consensus tree problem. In G.R. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *Lecture Notes in Computer Science*, pages 58–67, Berlin Heidelberg, 2005. Springer-Verlag.

100. D. Barker. LVB: parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, 20:274—275, 2004.

101. J.E. Gallardo, C. Cotta, and A.J. Fernández. Reconstructing phylogenies with memetic algorithms and branch-and-bound. In S. Bandyopadhyay, U. Maulik, and J. Tsong-Li Wang, editors, *Analysis of Biological Data: A Soft Computing Approach*, pages 59–84. World Scientific, 2007.

102. E.L. Lawler and D.E. Wood. Branch and bounds methods: A survey. *Operations Research*, 4(4):669–719, 1966.

103. P. Daniel and C. Semple. A class of general supertree methods for nested taxa. *SIAM J. Discrete Math.*, 19(2):463–480, 2005.

104. Alkim Ozäygen. Phylogenetic supertree construction using constraint programming. Master's thesis, The graduate school of natural and applied sciences, Çankaya university, 2006.

105. C. Cotta, A.J. Fernández, and A. Gutiérrez. On the hybridization of complete and incomplete methods for the consensus tree problem. *Manuscript in preparation*, 2008.

106. Francesca Rossi, editor. *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*. Springer, 2003.