MALLBA: Skeletons for Hybrid Methods in Combinatorial Optimisation

Alba E., Cotta C., Díaz M., Soler E., Troya J.M.
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
{eat, ccottap, mdr, esc, troya}@lcc.uma.es

May 5, 2000

1 Introduction

In its broadest sense, hybridisation refers to the inclusion of problem-dependent knowledge in a general search algorithm. Nowadays, it is a well-known fact that performing hybridisation is a must in order to guarantee a ground-level quality of the results [16].

As shown in the optimisation-related literature, there exists a enormous amount of techniques and mechanisms for carrying out hybridisation [8, 7, 15]. Nevertheless, it is possible to identify some common underlying principles that allow classifying these mechanisms into two major categories [3]:

1. *Strong hybrids*: algorithms in which knowledge has been included as specific non-conventional problem representations and/or operators.
2. *Weak hybrids*: algorithms resulting from the combination of lower-lever hybrid algorithms.

Besides theoretical disquisitions assigning the same power to both classes [3, 5], weak hybridisation can be approached as a "plus" for strong hybridisation, i.e., as a tool that can be put on top of some preexisting strongly hybrid algorithms to improve their performance. This is the kind of hybridisation in which we will concentrate.

The remainder of this work is organised as follows: first, we present some of the optimisation problems we are interested in (Sect. 2). Then, some general considerations about the design of skeletons for weak hybrid methods are introduced (Sect. 3). Subsequently, a proposal for hybrid skeletons is outlined (Sect. 4) and exemplified (Sect. 5).

2 Problems

This section will describe in some detail the opimisation problems on which we will focus. These problems are intended to be of practical importance and well-suited for the combination of different search techniques.

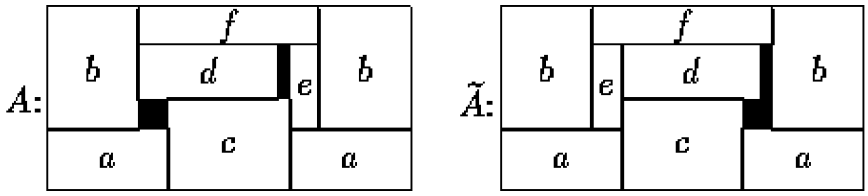

Figure 1: Examples of Rectangular Cutting

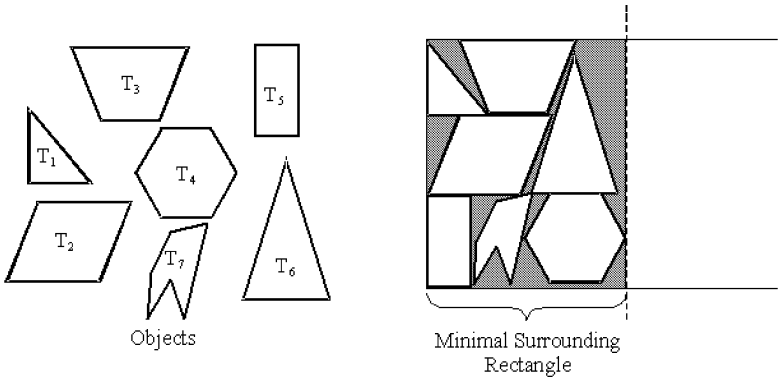

Figure 2: Generalised Cutting Problem

2.1 Optimal Cutting Problems

Optimal cutting problems are defined as follows:

Let $T = \{T_1, \cdots, T_n\}$ be a set of n objects such that every $T_i \subset \mathbb{R}^2$ is a closed, connected and bounded region. These objects can be translated but no rotated. The objective is finding an allocation of objects in T, such that no T_i and T_j overlap and the area of the minimal rectangle that contains all objects is minimal.

Because of their practical importance, a lot of work is done with respect to modeling, exact and heuristic solutions. In general, optimal cutting problems belong to the NP-hard class, but in many cases problems of medium size can be solved to optimality at an acceptable cost, especially when all objects are rectangular (see Fig. 1).

State-of-the-art algorithms for solving optimal cutting problems include evolutionary algorithms (EAs) [2] and simulated annealing [10] on the heuristic side, and branch and bound [11] with Lagrangean-relaxation techniques on the exact side. The existence of both approaches, heuristic and exact, makes this problem be very interesting from several perspectives, including benchmarking and the design of hybrid approaches.

Real-world instances of optimal cutting problems are available. These correspond to a textile manufacturer located at Málaga, and involve highly complex, non-convex objects (see Fig. 2). Currently, the manufacturer approaches this problem by using human operators, and is very interested in adding automatic functionality to the process, even allowing human operators for doing final refinement of the results.

2.2 Radio-Link Frequency Assignment Problems

The goal in this problem is to make an assignation from a set of frequency values to a set of radio links. This assignation must satisfy a set of constraints relating the number of different frequencies used, internal domain constraints for every link, and a group of equality and inequality equations among the frequencies assigned in the solution.

Formally, each link $i \in L = \{1, 2, 3, \cdots, n\}$ is assigned a frequency value f_i from a domain D_i, which is a subset of the set of all available frequency values. Typically, the domains have non-null intersections. In one of the most widely used data sets (CELAR) there are 7 domains, each containing between 6 and 50 frequency values.

For the particular problem types we are studying, there exists a large number of interference constraints of the following types: for a given pair of links i and j, it is necessary to satisfy that $|f_i - f_j| < \epsilon_{ij}$ or $|f_i - f_j| = \epsilon_{ij}$ for some frequency separation ϵ_{ij}. In the CELAR data set there are between 200 and 500 equality constraints and 1000 to 5000 inequality constraints. The first type of constraint is enough to make the problem NP-hard.

A given assignment is feasible if it satisfies all given constraints. An optimum is a feasible assignment that minimises some objective function such as the distinct number of frequency values used. Some problem instances have no feasible solutions. In such cases we should find an assignment which satisfies as many constraints as possible. In some problems, each constraint is given a value corresponding to its importance. The optimised function should ideally include terms which give a higher weight to the higher priority constraints.

The RLFAP is easily known to be NP-Complete and generates an enormous solution space. For example, a particular problem with 50 frequency values and 200 links (the smallest problem in CELAR) has an unconstrained search space of 50^{200}. The CELAR Problem 2 has 200 variables and 1235 constraints. It is known to have many feasible solutions and the optimum number of distinct frequencies is 14.

Multiple heuristic and exact algorithms can be used to solve this problem. Besides, hybrid combinations can be employed to solve instances drawn from real world benchmarks. Some of these algorithms are evolutionary algorithms, tabu search, branch and bound, and other techniques.

2.3 Coalition Formation in Multi-Agent Systems

Coalition formation is a key problem in multi-agent systems. The goal in this problem is finding a coalition structure maximising the value of the compounding coalitions. This is difficult to achieve in practice because of the huge number of potential structures.

In many domains, self-interested agents can obtain a high profit by coordinating their activities with other agents. Thus, it can be useful to automatise coordination activities by means of a negotiation software representing each of the involved parts. this software must carry on three activities:

- *Generation of the coalition structure*, i.e., finding a complete, non-overlapping partitioning of the set of agents.
- *Optimise each coalition*, i.e., assigning task to agents, taking into account resource constraints and maximising the resulting profit.
- *Divide the profit among the members of the coalition.*

These three activities are strongly interrelated. For example, the internal optimisation of coalitions depends on the coalition structure.

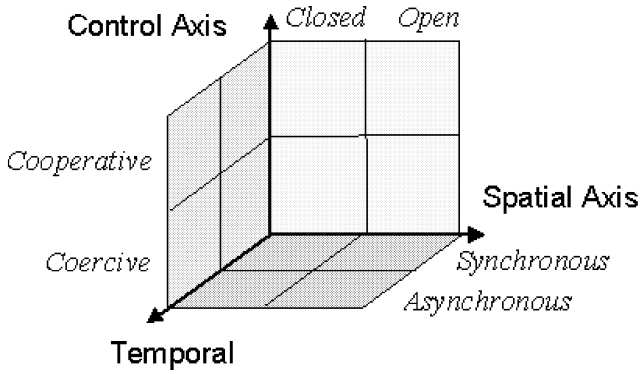

Figure 3: Reference frame for classifying weak hybrid models.

This problem is very similar to the *Set Covering* Problem, although it must be taken into account that determining the value/profit of a given subset is another optimisation problem itself. For this reason, this problem is very well suited for hybrid approaches: both exact and heuristic techniques have been applied to set covering. Furthermore, different techniques can be used to optimise the coalition structure and the internal assignment of tasks respectively.

3 Skeletons for Hybrid Algorithms

This section will provide some general considerations about the design of algorithmic skeletons for weak hybrid algorithms. In this sense, the structure of these algorithms is first studied. Then, issues regarding the implementation of such skeletons, e.g., interoperability, internal-state manipulation, etc., are analysed.

3.1 A Reference Frame for Weak Hybridisation

Weak hybrid algorithms can be classified using a three-dimensional orthogonal reference frame composed of a control axis, a temporal axis and a spatial axis. The first axis references the autonomy with which each algorithm performs its search. The second axis reflects the temporal aspects of the interaction between algorithms. Finally, the third axis corresponds to the limitations that each algorithm may have in its search space. When a weak hybrid model is mapped onto this reference frame, each coordinate may take two values as shown in Fig. 3.

Each of these axes is described below in more detail.

3.1.1 Control Axis

The control axis determines whether the two search algorithms interact at the same level or, on the contrary, they have a hierarchical relationship. The first situation corresponds to *cooperative* models in which each algorithm autonomously performs its search and occasionally interchanges information. The second situation describes *coercive* models in which an algorithm imposes how the other one must search.

Cooperative models involve techniques with similar behaviour (in terms of execution time and convergence speed), so they can effectively exchange useful information. Coercive models are usually related to the use of an auxiliary technique as an embedded operator (e.g., a mutation operator in a EA) that is given a starting point and/or a frame in which its search is to be done.

3.1.2 Temporal Axis

The temporal axis captures aspects concerning when interactions take place and which the behavior of each algorithm is between such interactions. Two options are considered: *synchronous* and *asynchronous* functioning. The first case includes those models in which an algorithm waits for the other one at a synchronisation point (often the termination of the latter) before continuing. The second case takes place when the algorithms do not wait for each other.

These two situations are very well exemplified in models that use a complex search technique as one of the internal search operators. The higher-level algorithm may invoke this operator and wait for the results (synchronous functioning) or continue its search, processing the results as they are received (asynchronous functioning). The second option may be appropriate when this hybrid operator is very complex and time-consuming [6]. Models in which the initialisation function uses solutions provided by another algorithm are clear examples of synchronous functioning too[1].

3.1.3 Spatial Axis

The spatial axis allows classifying weak hybrid models from the point of view of the search space each algorithm considers. Hence, *open* and *closed* models can be distinguished. No restriction is imposed on the search space in open models, i.e., each algorithm can theoretically reach any point of the search space. In closed models, the exploration is restricted to a certain subspace. Notice that this restriction is not related to feasibility constraints since infeasible solutions do not really belong to the search space in purity. On the contrary, this is a working characteristic of the algorithm, which is expected to perform better in this way.

Models performing local search via an improvement heuristic (e.g., hill climbing in Mühlenbein's parallel genetic algorithm [13]) are examples of open hybrid methods; any point of the search space could be reached after local search (of course, subject to the particulars of the fitness landscape [9]). On the other hand, the globally optimal forma completion described by Radcliffe and Surry [14] is a closed technique since exploration is restricted to a certain hyperplane of the search space.

3.2 Skeletons or Meta-Skeletons?

As mentioned above, there exist at least 8 different models for performing weak hybridisation. Moreover, each model admits a high number of variants depending on the particular search algorithms being hybridised. A question arises immediately: is it possible to define a general skeleton comprising all these variants? the answer is "no" and "yes".

We cannot define a general skeleton in the well-defined sense in which skeletons are defined for genetic algorithms, dynamic programming, etc. There is no underlying high-level common pseudo-code for all possible variants[2].

However, we can consider the possibility of defining a Meta-Skeleton for combining prefilt skeletons. To do this some requirements must be given:

- Each skeleton must offer a set of connectors to the exterior world. Each of these connectors will provide read/write/read&write access to two kind of internal objects:
  - Information objects: these objects represent the internal state of the algorithm. Their number and type are clearly dependent of the algorithm considered.

[1] In any case, issues regarding synchronism/asynchronism in LANs or WANs should be carefully studied [1].

[2] The term "high-level" is very important here. We could define a skeleton for a Turing Machine that obviously could be instantiated as desired. However we do not consider this possibility as of any utility for most users in general.

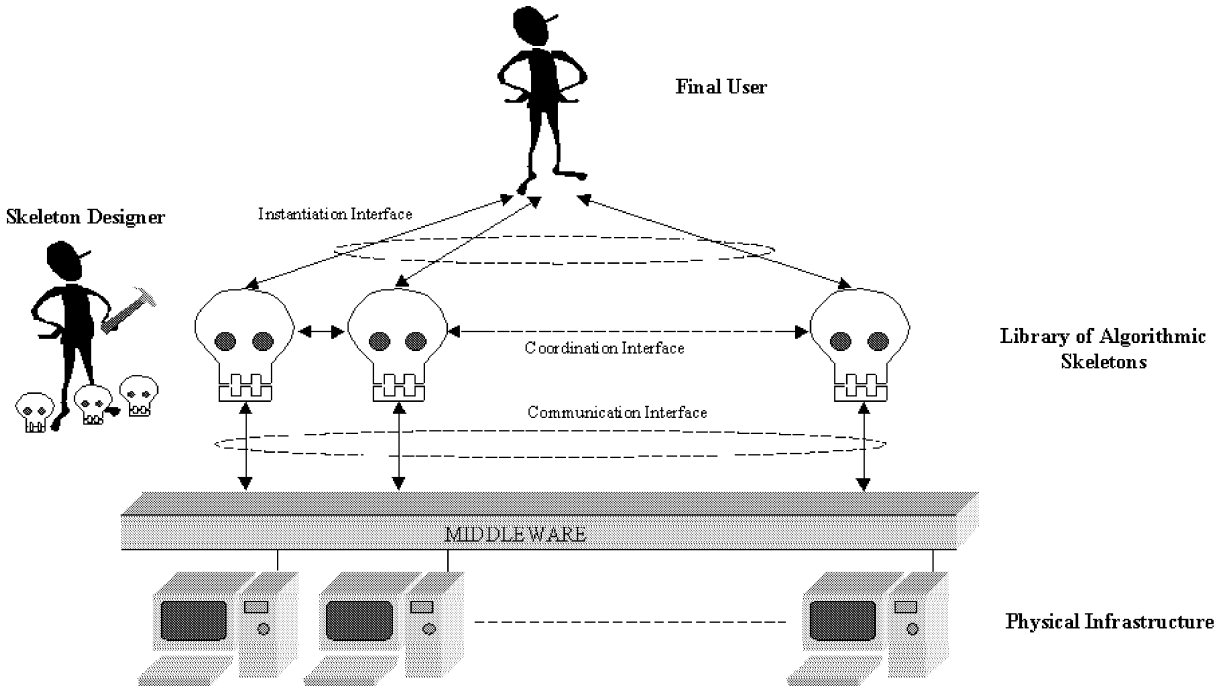

Figure 4: Architecture of the library.

- Control objects: these objects are used for modifying the internal execution state. There exist at least two major classes of control objects:
  - Synchronisation objects: intended for defining breakpoints, critical sections, etc.
  - Resume objects: intended for breaking the execution path of the algorithms. There always exist two of these objects: `restart` (resumes execution from the beginning) and `terminate` (finishes the execution of the algorithm).

- The user must specify two components of the meta-skeleton:

  1. The way in which the connectors of the algorithms combined are accessed and/or modified. This is the body of the meta-skeleton, e.g., expressed as an algorithm in C++.
  2. The connectors that the resulting hybrid algorithm will offer for subsequent hybridisation.

The definition and use of this meta-skeleton has advantages and disadvantages. On the positive side, it is very flexible and provides the user with full-power for defining all aspects of the interaction between algorithms. On the negative side, the design of such a system can be overwhelming for a novice user.

An alternative to meta-skeletons exists, i.e., defining as many skeletons as possible combinations of algorithms exists. Since this is clearly unaffordable, should this option be chosen the following steps would be required:

- Identifying the models to be implemented: already done in the project proposal, memetic algorithms [12] and dynastically optimal recombination [4].

- Coming up with a tradeoff between the flexibility/applicability of the skeleton and the complexity of filling it for a novice.

This alternative does simplify the design of the skeletons, but still some tools for controlling skeletons are needed, even when these are not offered to the final user but just to the skeleton designer(s).

To sum up, let us consider an updated version of the library architecture (Fig. 4) in which we have added a new character: the skeleton designer. The question is: which is the profile of the final user (or team of users)? More precisely, will it be close to the skeleton designer? Before answering to this latter question consider that it can be reformulated in two ways:

1. Will the skeleton designer play a prominent role as a final user? or
2. Will it be usual for the team of users in need of a weak hybrid to include a member with skeleton-design skills?

It is difficult to give the ultimate answer to this question. Nevertheless, it is important to consider that, whatever this answer is, there exists a common set of services that must be provided with independence of who will use them, i.e., the control objects plus access to the internal state of a skeleton. Hence, a working plan could be designing these services and use them initially just at the design level. A further optional step can be the design of a fully functional meta-skeleton.

4 A Proposal for the Inter-Skeleton Interface

Each skeleton must provide a method `GetState()` returning an instance of class `State`. This class handles all requests for consulting/modifying the internal state of an algorithm. A generic interface for this class would include:

- `Save(String filename)`: the state is stored in a file.
- `Load(String filename)`: the state is retrieved from a file. The algorithm can be subsequently restarted from this state.
- `ConnectorList GetConnectorList()`: this method returns a list of all connectors available in the state of the current algorithm, along with their type and access mode.
- `Bool AdquireConnector(String connectorName, ConnectorAccess accessType, Connector &connector)`: this method requests access (for reading, writing or both) to a connector.
- `ReleaseConnector(Connector c)`: this method releases access to a connector.

`Connector` is an abstract class including two virtual methods:

- `ConnectorType Type()`: this method returns the type of connector (`dataConnector`, `controlConnec`
- `ConnectorAccess Access()`: this method returns the allowed access mode (`read`, `write`, `readWrite`).

Now, `DataConnector` is another abstract class from which several classes can be derived: `IntegerDataConnector`, `RealDataConnector`,... Each of these will include methods for retrieving/setting the value of the connector.

On the other hand, `ControlConnectors` comprise `SynchronisationConnectors` and `ResumeConnectors` The former encapsulate typical structures such as locks, barriers, etc. As to the latter, they refer to certain execution points from which the algorithm can be forced to resume.

Every connector will be associated with an internal object. The number, type, access, etc. of this object are algorithm-dependent.

5 Example: Branch&Bound Connectors

Below is a possible list of connectors offered by a B&B algorithm (minimisation is assumed without loss of generality) :

- `initialProblem: IntegerListDataConnector // definition of the initial problem to be solved.`
- `currentUpperBound: RealDataConnector // cost of the best solution found so far.`
- `currentLowerBound: RealDataConnector // cost of the best subproblem in the queue.`
- `bestSolution: IntegerListDataConnector // best solution found so far.`
- `currentSubProblem: IntegerListDataConnector // subproblem to be examined.`
- `openSubProblems: QueueDataConnector // queue of pending suproblems.`
- `searchStrategy: IntegerDataConnector // strategy for examining pending subproblems.`
- `emptyQueue: LockSynchronisationConnector // synchronisation point at the end of B&B main cycle.`
- `restart: ResumeConnector // resume point: reinitialisation.`
- `terminate: ResumeConnector // resume point: termination.`

References

[1] E. Alba and J.M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 2000. forthcoming.

[2] Th. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[3] C. Cotta. *A Study of Hybridization Techniques and their Application to the Design of Evolutionary Algorithms.* PhD thesis, University of Málaga, 1998. In Spanish.

[4] C. Cotta, E. Alba, and J.M. Troya. Utilising dynastically optimal forma recombination in hybrid genetic algorithms. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 305–314. Springer-Verlag, Berlin, 1998.

[5] C. Cotta, E. Alba, and J.M. Troya. On the computational power of adaptive systems. *Computers and Artificial Intelligence*, 19(2), 2000. Forthcoming.

[6] C. Cotta, J.F. Aldana, A.J. Nebro, and J.M. Troya. Hybridizing genetic algorithms with branch and bound techniques for the resolution of the tsp. In D.W. Pearson, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 2*, pages 277–280, Wien New York, 1995. Springer-Verlag.

[7] 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-Verlag.

[8] H.-L. Fang, P. Ross, and D. Corne. A promising hybrid ga/heuristic approach for open shop scheduling problems. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 590–594. John Wiley and Sons, 1994.

[9] T.C. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, 1995.

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

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

[12] P. Moscató. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report #826, Caltech Concurrent Computation Program, 1989.

[13] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421, San Mateo, CA, 1989. Morgan Kaufmann.

[14] N.J. Radcliffe and P.D. Surry. Fitness variance of formae and performance prediction. In L.D. Whitley and M.D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 51–72, San Mateo CA, 1994. Morgan Kauffman.

[15] J.M. Varanelli and J.P. Cohoon. Population-oriented simulated annealing: A genetic/thermodynamic hybrid approach to optimization. In L.J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 174–181, San Mateo CA, 1995. Morgan Kauffman.

[16] D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.