Towards an on-line optimisation framework

EOLE[1] consortium - eole@thalesgroup.com
THALES Research and Technology: Simon de Givry, Youssef Hamadi, Juliette Mattioli, Philippe Gérard
ONERA / DSCD: Michel Lemaître, Gérard Verfaillie
COSYTEC: Abderrahmane Aggoun, Idir Gouachi
BOUYGUES / e-lab: Thierry Benoist, Eric Bourreau, François Laburthe
Ecole des Mines de Nantes: Philippe David, Samir Loudni
France Télécom / R&D : Serge Bourgault

Abstract

This paper is a survey and concept paper, listing and discussing various ideas relevant to the use of on-line solving. We propose a new software architecture for the design of on-line search algorithms. We define the temporal control, the main component of this architecture, for the case of partial search algorithms, well-suited to a limited response time. We also investigate hybrid search algorithms, which include partial search algorithms. Our main goal is to extend the Constraint Programming paradigm to on-line combinatorial problem solving. The extensions concerning the constraint solver are discussed in the last section.

1 Scope and objectives of the EOLE project

Optimisation functions, and quite particularly on-line optimisation functions, will be a major vector for the Quality of Services enhancement, by increasing flexibility and capacity of adaptation of Telecom systems to the evolution of the demand and to the state of their appropriate resource. The purpose of the EOLE project is to build an on-line optimisation framework dedicated to the Telecom domain, able to take into account environment events, reconfiguration possibilities, temporal constraints and resource constraints.

1.1 On-line optimisation problematic

Few years ago, Constraint Programming (CP) was identified as a key technology that was going to increase the efficiency and the competitiveness of companies. This technique, arisen from the interaction between Artificial Intelligence and Operations Research, opened perspectives of optimisation for numerous industrial applications. Ten years later, the CP contribution, for the expression and the resolution of complex combinatorial optimisation problems, is not to be any more demonstrated. The success emanates mainly from the flexibility that CP offers in term of modelling. CP is a paradigm that, not only allows solving combinatorial problems, but also is able to express them in their global nature by means of a unique formalism. It is not generally the case of the alternative techniques. This trump card turned out determining in the case of the industrial applications for which the specifications are often diverse, badly known at first, and subject to fast evolutions.

It is today indisputable that CP answers the industrial needs of optimisation off-line as a batch process. Numerous systems using this technology are already operational in different fields of activity: management, production, transport, and finance. However, these systems, to remain relevant in particular in the Telecom world, should become more and more dynamic and support on-line interactions with their environment. The solution should be maintained up to date dynamically according to the modifications of the environment or the available resources. These dynamic systems, subject to temporal constraints, do not benefit for the moment from the constraint technology because CP is well suited to the resolution of off-line problems. Not only no time or space guarantee is supplied, but also, the current CP solvers do not lend themselves to the combination with the other paradigms of resolution, such as stochastic programming (for example, see [Voudouris et al. 01]), better adapted to an on-line context.

1.2 EOLE objectives and work plan

In the current situation, a software engineer who wants to use CP for solving an on-line combinatorial problem has no tool at his/her disposal. Its unique choice is to interrupt the resolution at the end of the timeout, what is not always the best option in term of quality and turns out even sometimes impossible because no solution has been found. Our proposal is to build a CP system that selects and configures a search algorithm for each solving request. The algorithm seeks to produce a good solution based on some optimisation function. This system can be viewed as a purely reactive system. The proposed approach consists of combining the flexibility of modelling

---

[1] The French EOLE project, the name of which stands for *Environnement d'Optimisation en LignE* (on-line optimisation framework), is in part funded by the French *Réseau National de Recherche en Télécommunications*. See http://www.lcr.thomson-csf.com/projects/www_eole.

of constraint technology, the power of resolution of hybrid algorithms, and the relevance to real-time operational constraints of anytime algorithms[2]. The evaluation of the quality of a solution, the temporal control of search algorithms, the search procedure parameterisation and the hybridisation of tree search methods and local search methods are the fundamental points that will be investigated by the EOLE project. Robustness or stability of the solutions are not studied.
From the point of view of the software engineering, the proposed framework will be developed under the shape of an on-line optimisation framework. This framework will offer, in the field of telecommunications, a set of dedicated customisable hybrid algorithms, as well as mechanisms allowing to compose them and to manage them according to the evolution of the situation and to the pre-established strategies of adaptation. This framework should contribute to increase the robustness and the capitalisation of the software optimisation inside the systems of telecommunication.

The EOLE project is a RNRT[3] project supported by the French Research Office. It began on April 2000, and it will be finished at the end of 2002. The work plan is divided into six sub-projects. The first one consists in establishing the foundations, concepts and perimeter of use of an on-line optimisation framework. This sub-project should supply with formalisms to express algorithms the behaviour of which answers the defined characteristics. The second sub-project consists in offering above the Claire© language, and above the ECLAIR© constraint library[4] or the Choco constraint library, primitives implementing the previous formalisms. Finally, it aims at producing the integration of these tools and libraries in an object-oriented framework. The following three sub-projects allow the validation through a set of benchmarks led on three experimental applications:

- The network allocation optimisation of resources during a reconfiguration,
- The ATM networks management,
- The frequencies assignment for the cellular phone.

And the last sub-project attempts to define an industrialisation plan of the developed technology.

2 A generic architecture for on-line constraint optimisation

The EOLE architecture described here is the first CP attempt to efficiently address on-line applications. This framework will consider the environment of the systems, the needs for reconfiguration, temporal and resources constraints. It will be the first integrated system dedicated to telecom applications test and development. In this system, versatility is used to face dynamic changes of the environment. Search procedures are becoming adaptive and temporal constraints are used to guide the adaptation in the on-line framework. In this section, we describe the different components of the system. To face this important challenge we use state of the art research results joined to pragmatic software integration. In the following, we show that decomposition combined to well defined relations are the keys to reach reactivity in an on-line framework.

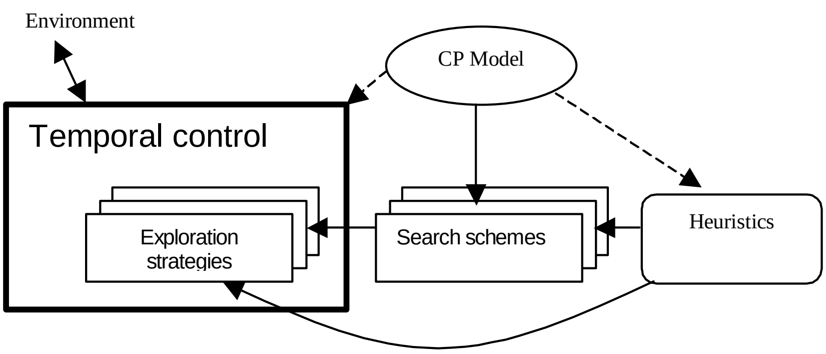

Figure 1: the EOLE architecture.

The figure 1 presents the EOLE architecture. This architecture improves the basic conceptual presentation of on-line optimisation architecture. Usually these descriptions are very high level. They distinguish between a decisional module and a solver. These two components interact by displaying results (from the solver) in response to a previous command (from the decision module). Our architecture refines this classical representation. In the following we describe the different components and their relationships.

[2] Anytime algorithms are algorithms whose quality of results improves gradually as computation time increases.
[3] RNRT: Réseau National de Recherches en Télécoms
[4] Claire© copyright Yves Caseau ; ECLAIR© copyright THALES Research and Technology

2.1 Model

The model is a classical CP model. It defines an optimisation problem by using variables with sets of values and by restricting combination of values by the way of constraints. Optimisation variables are defined within the model.

2.2 Heuristics

The first goal of the "Heuristics" component is to guide the resolution strategies. It can also be selected to enhance search strategies. Each heuristic uses a normal representation. This allows easy comparisons as well as efficient combinations. A combination of heuristics acts as a parameter addressed to a search scheme or an exploration strategy. The link between the model and the heuristic component allows heuristic instantiation [Minton 96].

2.3 Search schemes

A search scheme defines a structured view of a search space. We can define both local search and tree search spaces. Tree search is implemented by defining an ordered set of choice-points. Each choice-point refers to a sub-set of variables presented in the CP model. Local search will refer to these elements by defining particular evaluation function and neighbourhoods. A combination of choice-points defines a complete search-space. As presented above, a search scheme can refer to variables/values heuristics.

2.4 Exploration strategies

The EOLE architecture confronts between search-space and search-space exploration. This division allows a distinction between a search procedure and its topic. An exploration strategy is applied on a search-space defined by a search scheme. This separation gives efficient code reusing within the framework.
Exploration strategies try to solve a problem by doing an efficient exploration of its search space. They relate choice-point and limitations in the exploration. More generally they give for each choice-point the right focalisation by cutting the set of alternatives (e.g. set of values for a variable). These limitations allow us to efficiently implement classical explorations (LDS, Iterative Broadening, Credit/Barrier, etc.). Local search methods can also benefit from the limitations (max-flips, max-tries, etc). By switching between these strategies, the EOLE architecture automatically defines and uses hybrid search. The connection between these strategies and the heuristic box represents possibility for the user to change elements of the propagation engine (shaving, fifo/lifo, hierarchical propagation, etc.).

2.5 Temporal control

The temporal component is the heart of the EOLE architecture. This component is connected to the outside world. It receives requests and presents results to the end user. Moreover it guides the adaptation of the running strategies to fit temporal and resource constraints. This component uses two operating mode corresponding to possible requests from the outside world.

2.5.1 Contractual mode

The system receives resolution requests with a temporal constraint. Several tools are available here to select the right search strategy. For example, the connection with the model allows prediction by the exploitation of structural information (phase transition localisation, etc) [Cheeseman 91]. Learning about previous resolution can help to find the best strategy. Interestingly, the on-line feature of the system allows efficient and cheap prediction and learning since each solved instance can be used to enhance the system's knowledge. With this knowledge, the temporal component can adjust the limits of a search strategy. This correction corresponds to the choice of a particular strategy. In this mode, the system can chain different search if the temporal constraint is not exhausted.

2.5.2 Interrupting mode

The environment addresses a resolution request that can be interrupted at any time. The temporal module will successively consider artificial temporal constraints. These constraints range from the tightest to the weakest. Each constraint will be used to select the right search strategy. By this way, the system will always be able to propose a solution to the problem and more importantly it will continue to refine it. This mode looks like a succession of contractual executions. But interestingly, between each resolution, the previous solution can be used to enhance reasoning of the following search.

The different components described here are the basis of an on-line optimisation framework. In the next section, we refine this architecture to the case of partial search. Our final goal is to build a set of high-level search primitives related to each functional component. These primitives help a technical expert for the design of on-line optimisation algorithms. Section 4 proposes a set of primitives for hybrid search algorithms.

3 Temporal control of partial search algorithms

3.1 A survey of estimation tools

Controlling a search algorithm implies to answer general questions such as:

- How long will the algorithm take to finish the search?
- How far is the current best solution from the optimum?
- What will the improvement of the solution quality be if the search continues for a given time?

These questions are in general difficult to answer, due to the complexity of combinatorial problems and the non-determinism of search algorithms. We present the existing solutions based on estimation only.

3.1.1 Estimating the search time

Estimating the time of a complete search enables to know how long it takes to find an optimal solution or a solution with a given level of quality. We distinguish three approaches to estimate the time of tree search algorithms:

- Statistic models of search trees. [Pemberton & Zhang 96] give a simplified model of a search tree, which is defined by its maximum depth, its average branch factor and the distribution of the evolution of the objective cost along a path. This model gives an analytical estimate of the number of nodes of the tree. If we know the time needed to explore a node[5], we obtain an estimation of the time required to explore the tree. But we don't know whether the simplified model reflects the reality.
- The *iterative sampling method* [Knuth 75] consists in performing several random descents along the paths of a search tree. Collecting the depth of the descents and the branch factors observed along these paths, we can deduce an estimate of the number of nodes of the tree. In the case of minimisation problems, the method does not evaluate the size of the actual tree, but the size of an approximate tree, which corresponds to the number of solutions with a cost lower than an initial upper-bound. The estimator variance can be estimated. Experiments show that the variance is generally very high. The advantage of the method is its ability to adapt to all combination of problem instances and algorithms. Moreover, it can be used during the search, to estimate the size of the remaining part of the tree.
- The *ratio method* is a simplification of the above method. It consists in running the tree search algorithm for a short period of time. From the ratio between the size of the explored part of the tree and the whole tree[6], we simply deduce an estimation of the total search time. This can be applied at any time during the search. This method makes the assumption of a homogeneous tree.

All these methods lack accuracy, especially if the search tree is not well-balanced. They apply only to basic complete search methods. In the case of *partial search*[7], these methods have to be adapted. For example, a limit on the number of nodes of a sub-tree is an upper bound for the *iterative sampling method* or the *ratio method*. Estimation methods for *partial search* should give better results than for complete search if the search limits are strong (the size of the tree becomes relatively small).

The time for estimating the duration of a search is an important criterion to select the right estimation method. [de Givry et al. 99] dynamically choose the parameter values of a partial search algorithm, such as a limit on the branch factor, according to a given time limit and a given problem instance. A fast estimation based on the *ratio method* permits to quickly assess different parameter values. This assessment is done regularly, which counterbalances the relative inaccuracy of the ratio method.

Note that in the case of local search, time estimations are easier. For example, for GSAT [Selman et al. 92], it depends directly on the number of tries and the number of flips. In the case of hybrid search, it gets more complex. Hybrid search generally involves a number of internal local and tree searches. A specific estimation should be done for each search. It may not be possible to do that before running the hybrid search because the searches are often interdependent.

Another approach, which always works, consists in doing off-line experiments on a set of problem instances. It is also useful for other estimation purposes, and it is discussed in the section 3.1.3.

---

[5] The actual time is subject to high variations due to the propagation of global constraints.

[6] Multiplying together all the sizes of the choice points belonging to the current path when the search stops approximates the size of the whole tree.

[7] A partial search is a tree search with some limits on the number of visited nodes. For example, one iteration in the Limited Discrepancy Search algorithm [Harvey & Ginsberg, 1995] is a partial search. See section 3.2.1 for more details.

3.1.2 Estimating the distance from the optimum

It is not possible to know the optimum before the end of a complete search, but we can bound the optimum by a lower and an upper bound. The difference between the upper bound and the lower bound gives an upper bound on the distance to the optimum. The distance estimation is a way of assessing the quality of a given solution. In the case of a minimisation problem, search algorithms applied to the whole problem naturally produce upper bounds. It is also the case if we constrain the problem further in order to obtain an easier problem (for example, a polynomial problem). Lower bounds are obtained by:

- The initial propagation on the variable corresponding to the objective function.
- A complete Branch and Bound algorithm applied on a limited tree search, or using a best-first strategy. An idea is to use the lower bounds, which are computed at each node of the tree, in order to produce a global lower bound. See [Cabon et al. 98] for experiments on six variants of a Branch and Bound search, producing anytime lower bounds.
- Solving completely a simplified problem: the optimum of the simplified problem, either is a lower bound of the original problem, or allows computing such a bound. The simplification may be a relaxation of the constraints, such as removing or weakening some constraints. It may be a modification of the objective function [de Givry et al. 97].
- Studying the structure of the minimisation problem, one can deduce obvious lower bounds. For example, in the Travelling Salesman Problem, a trivial lower bound is the sum of the shortest distances starting from every city.
- For some combinatorial problems, there exist approximate algorithms that guarantee a distance to the optimum (see [Cormen et al. 94] for an example on the Travelling Salesman Problem).

Except for the first two approaches, the ways of producing lower bounds are problem dependent.

3.1.3 Estimating the evolution of the solution quality

Knowing the evolution of the solution quality is a key point for controlling search algorithms. The estimation approach is based on performance profiles studied in Anytime Algorithms (see for an overview, [Zilberstein 96]). A performance profile of a search algorithm, $Q(t)$, denotes the expected solution quality with execution time t. Performance profiles are constructed off-line empirically by collecting statistics on the evolution of the solution quality of a search algorithm on many problem instances.

There are several difficulties to obtain relevant statistics:

- The problem instances used for the experiments must be representative of the real instances.
- The problem instances have to be characterised by some parameters in order to classify them into families having the same performance profile. These parameters may be the instance size (e.g. number of variables), the intrinsic difficulty (e.g. tightness of the constraints), etc.
- The solution quality should be normalised. This is not true if the solution quality derives directly from the objective function. An idealistic approach would be to compute off-line, for each problem instance, its worst solution cost, W, and its optimum, O. A normalised quality is $Q_{norm}(t) = (W - C) / (W - O)$, where C is the cost of the best solution found at time t. A pragmatic approach is to define the quality as the percentage of improvement with respect to the initial cost, C_0, of the first solution found by the search algorithm, $Q_{pragm}(t) = (C_0 - C) / C_0$. This pragmatic approach gives an estimation of the evolution of the solution quality. Other approaches have to be defined if we want to use the performance profile in a different goal, such as the comparison of several algorithms.
- The experiments should cover all the problem instances and all the available search algorithms. The execution time of one search has to be long enough to cover any time limit.

The effort spent on the experiments will improve the quality of the performance profiles, by reducing the estimate's variance. The experiments may be too long if the problem instances are too large, if the maximum time limit is too long, or if we test too many search algorithms. Hopefully, in an on-line context, the successive problem instances to be solved evolve progressively. Successive instances have quite the same difficulty. This regularity enables to enhance the estimation by using some learning mechanisms.

3.2 Overview of partial search algorithms

In this section and the next one, only tree search algorithms and their temporal control are discussed. We focus our work on tree search methods because of their compatibility with the Constraint Programming paradigm. We call a partial search algorithm any tree search algorithm that has been designed for a context where the running time is limited. [Harvey 95] calls it *Nonsystematic Backtracking Search*. The main idea is to diversify the search by avoiding the trashing phenomenon of systematic backtracking methods[8]. The diversification principle is also applied in local search methods for escaping from local minima. Partial search consists in restricting the search

[8] Systematic backtracking methods can spend a very long time to explore a sub-tree containing no feasible solution or only sub-optimal solutions. See [Hogg et al. 96, Gomes et al. 98] for an analysis of this phenomenon.

by applying limits on the explored nodes and in modifying the exploration order of the nodes by using the heuristics in a more clever way. We call these limits, the *search limits*. We give a classification of the most promising existing partial search algorithms:

- The **Iterative Weakening** methods. Typical examples are Iterative Broadening [Ginsberg & Harvey 92], Limited Discrepancy Search [Harvey & Ginsberg 95], etc. The strategy is to solve the same problem repeatedly with the search limits progressively relaxed at each iteration. The successive searches are of increasing complexity, until optimality is proved or the deadline is reached.
  The main drawback of these methods is that each new iteration revisits all the interior[9] nodes of the previous iteration, except when a tighter upper bound has been found. The ability to find decreasing upper bounds quickly during the first iterations helps the Depth First Branch and Bound algorithm to cut branches earlier in the next iterations. This point alone justifies iterative methods.
- The **Real-Time Search** methods. The idea is to adjust the size of the explored search tree to the time limit. The search must end close to the deadline. [de Givry et al. 99] use time estimation in order to select the right parameter values: regularly, a controller estimates the time to finish the search with the current parameter values and compare it with the remaining time. If the difference is too large, the parameters are tuned again. [Korf 90] dynamically adjusts the depth of a look-ahead search, inserted in a *Best First Search* algorithm. [Chu & Wah 91] dynamically adjust the convergence speed parameter of the *Iterative Approximating* algorithm.
- The **Iterative Sampling** methods. They consist in trying several different value and variable heuristics rapidly by doing greedy searches or very incomplete searches. A way of obtaining new heuristics consists in randomly biasing a given heuristic (see [Gomes 98] for biased variable heuristics and [Bresina 96] for biased value heuristic). A simpler approach consists in using random value selection (see for example [Langley 92]). The main drawback of these methods is a difficulty to improve the solution quality when a large amount of time is allocated. This is due to a large degree of incompleteness and also to blind searches in case of a random selection. A solution proposed by [Gomes 98] is to increase the search complexity every n searches. The authors show that multiple searches eliminate the phenomenon of "heavy-tailed cost distributions". This means that at any time during the experiments, we have a non-negligible probability to encounter a problem that requires exponentially more time to solve than the previous ones. They also show that multiple searches have a non-negligible possibility of very short runs that dramatically shorten the solution time.
- The **Interleaving** methods. This approach consists in solving simultaneously different parts of a single search tree. In practice, the tree is divided into several sub-trees starting from the root. The sub-trees are explored by using an interleaving approach [Meseguer 97]. The idea is to reduce the cost of heuristic errors at the top of the tree. Parallelism instead of interleaving is easy to implement and can provide super-linear speedups.

3.3 The contractual mode for partial search algorithms

In a contractual mode, the time limit is known before the search. Our objective is to take advantage of this information as much as possible. We describe a new methodology for designing partial search algorithms that take into account a limit on their running time. This design methodology corresponds to an instance of the generic architecture given in section 2. The temporal control of partial search methods is defined explicitly, as explained below.

3.3.1 Parameterised tree search algorithms

In a limited time, partial search algorithms explore a small fraction of the search tree only. Our goal is to express explicitly what nodes should be visited. There are multiple ways of limiting the search. Assuming a standard depth first branch and bound algorithm, we suggest three main approaches:

- **Limits related to a node**. For a given variable, restricts the value enumeration to a subset of the domain according to some heuristic. For example, uses a cardinality condition (n best ranked values) or a metric condition (maximum distance to the best value). Or uses a condition on the node's lower bound by enforcing arbitrary smaller upper bounds[10]. For example, replaces the backtrack condition of the classical Depth First Branch and Bound ($ub \le lb(node)$) by a weaker condition ($\varepsilon \,.\, ub \le lb(node),\ 0 < \varepsilon \le 1$).
- **Limits related to a path**. For example, for a path involving a set of variables, restricts the cumulated number of heuristic discrepancies [Harvey & Ginsberg 95]. Or uses a condition on the node's lower bound, enforcing $lb(current\_node) \le lb(father\_node) + \delta$ (see [Zhang 98]).

[9] Some iterative methods do not revisit the leaf nodes of the previous iteration [Korf 96, Walsh 97].
[10] For minimisation problems.

- **Limits related to a sub-tree**. For example, uses some topological conditions (maximum number of nodes, maximum number of backtracks, maximum number of leaves, etc.). Or uses a global metric condition (*n* best leaves).

We can combine the limits. In [Beldiceanu et al. 98], a discrepancy limit and a maximum number of backtracks are applied. Also, we can restrict the scope of the limits to a given depth interval in the search tree.

The above examples exhibit several parameters, which control the complexity of a search. Our objective is to produce from a single scheme a parameterised search algorithm that lies somewhere between the greedy and complete extremes. This allows to reuse the same algorithm with different parameter values for different time limits. The search limits correspond to an exploration strategy in the EOLE architecture.

3.3.2 How to control one search?

We distinguish three different *tuning strategies* for tuning the parameters of a parameterised search algorithm, depending on the dynamics of the strategy:

- A **static tuning** of the parameters. It consists in fixing the parameters before the search so that the algorithm performs well on average for a set of problem instances and a given contract duration. To achieve that, we use the performance profiles obtained by off-line experiments to choose between several parameter configurations of the search algorithm. The main drawback of problem-dependent tuning is the inability to adapt to a particular problem instance. For example, the algorithm can terminate long before the end of the contract duration or can be interrupted long before its termination.
- An **iterative tuning**. This is the strategy applied by *Iterative Weakening* methods. The iterative methods can be seen as a way of transforming a contract algorithm such as a parameterised tree search algorithm into an anytime algorithm. A simple strategy, optimal under certain assumptions, is to double the search complexity at each iteration (see for a proof [Russell & Zilberstein 91]). This can be done by weakening the search limits of the next search such that the estimated time of the next search doubles the time of the previous search. And during the last iteration, a further improvement consists in applying a dynamic tuning to adjust to the remaining time.
- An **adaptive tuning**. Here, we want to adapt the search parameters to the specific instance to solve. A way to do that is to collect information during the search in order to refine the parameters of the instance and to reuse the corresponding performance profiles to choose the best tuning. This approach is applied in the Eureka system [Cook & Varnell 97]. A different approach is to adjust the size of the explored search tree to the time limit like it is done in *Real-Time Search* methods.

In the case where the search limits are tuned dynamically, we distinguish two situations:

- The search limits are tightened. The size of the explored search tree is reduced.
- The search limits are relaxed. The size of the tree grows. In the case of a depth-first search, the sub-trees already cut by the previous limits will not be explored in spite of the enlarged limits. We have to restart the depth-first search to be able to explore these sub-trees.

Note that a dynamic change of the time limit (due to some external events) can be handled similarly: the parameters have to be tuned again.

What tuning strategy should one choose? It depends both on the time limit and on the problem. Note that a static tuning can be combined with an adaptive or an iterative tuning, by setting some parameters.

In the three approaches described above, we can simplify the tuning process by providing an ordered tuning policy. The number of possible values for the parameters may be huge. It is impossible to test all the combinations. Also, some different combinations may imply the same search duration. Thus, we propose to establish experimentally a list of interesting tuples of parameter values. Then, we can sort these tuples by an order of increasing complexity, the first tuple corresponding to a greedy search and the last tuple to a complete search. We call this ordered list a *tuning policy*. Now, the search limits can be tuned by a unique global parameter, which is the index in this list. Modifying the index has a direct effect on the search complexity and the solution quality. Such a tuning policy is important to guide the iterative tuning (especially in the case of several limits) and to speed up the convergence of the adaptive tuning.

The figure 2 summarizes our methodology in terms of architecture for expressing a parameterised tree search algorithm and its tuning policy. The tuning strategy is chosen by the temporal strategy, presented in the next section.

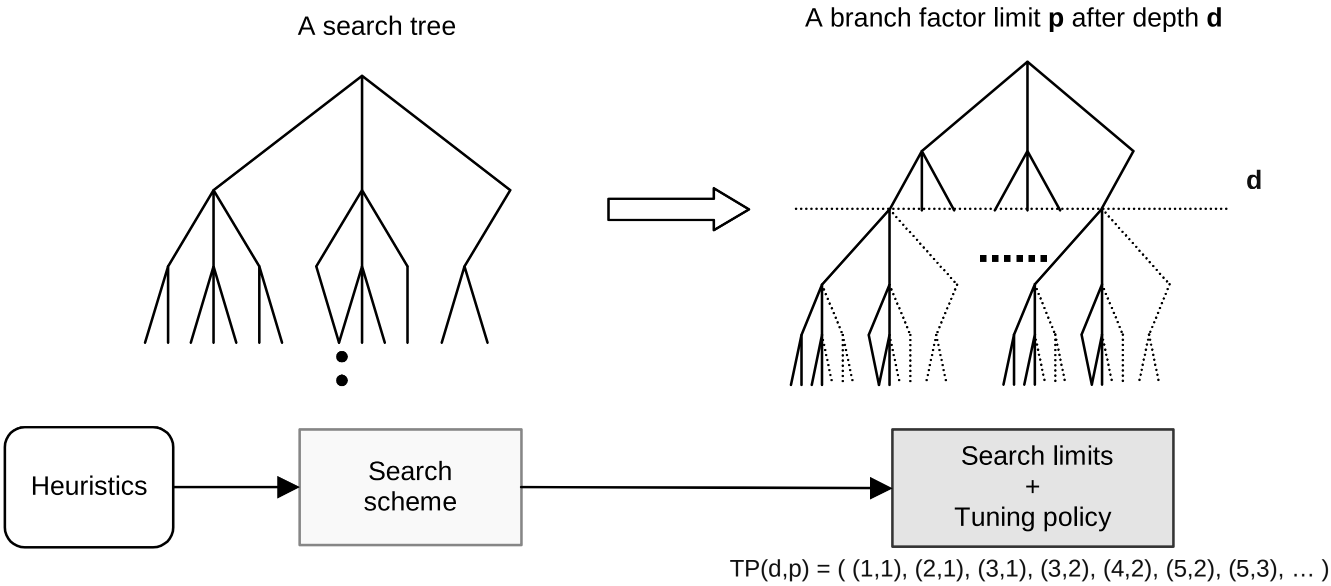

Figure 2: Architecture of a parameterised tree search algorithm defined as the application of search limits to a search scheme, with a tuning policy indicating the permitted values of the limits. In the example, the first element of the tuning policy is (d,p) = (1,1) corresponding to a greedy search. The tuning policy increases the backtracking at the root of the search tree more than at the bottom.

3.3.3 How to control multiple searches?

In the case of the iterative sampling methods and the interleaving methods, they perform several searches with different search schemes or different heuristics, either sequentially or by interleaving. We propose two operators to specify the *temporal strategy* of multiple search methods:

- `seq`, the sequence operator (`S1 seq S2`). The searches are performed one after the other.
- `int`, the interleaving operator (`S1 int S2`). The searches are performed simultaneously (by interleaving or in parallel): the computational resources are shared by all the search processes.

The searches just share the best solutions found so far. The current best cost is used as a cut in every search.

The `seq` and `int` operators express a basic temporal control of a complex search algorithm: they specify how to allocate the available time to each search. By default, in the case of a sequence, the total time is given to the first search, the remaining time is given to the second search, and so on. In the case of interleaving, the same time is allocated to every search, and the computational resources are shared equitably.

We introduce the notion of *time-sharing policy* to express more clever temporal strategies. We add such a policy to the definition of a sequence or an interleaving. For example, for a sequence of *N* searches, the time limit can be divided into *N* parts, with a fixed search ratio for each part. In the case of interleaving, the most promising searches can get more computational resources as time passes. [Prcovic & Neveu 00] choose to allocate more cpu time to the search that reaches the deepest node in the search tree.

The temporal control selects[11] and monitors a temporal strategy. The temporal strategy defines a hybridisation process explicitly (the control of different searches). It stipulates how to distribute a given contract to a number of searches. Each search will have its own deadline used by its tuning strategy (see section 3.3.2), which is chosen by the temporal strategy. In our opinion, this is a major concept for the design of on-line search algorithms.

The figure 3 shows the architecture for expressing a partial search algorithm taking into account a time limit. The figure 4 derives the EOLE architecture to the case of a contractual mode for partial search algorithms. And finally, the figure 5 summarizes how a time contract is taken into account in our approach. There is a THALES patent related to all the architecture described in section 3.3.

[11] Based on some selection rules obtained by learning.

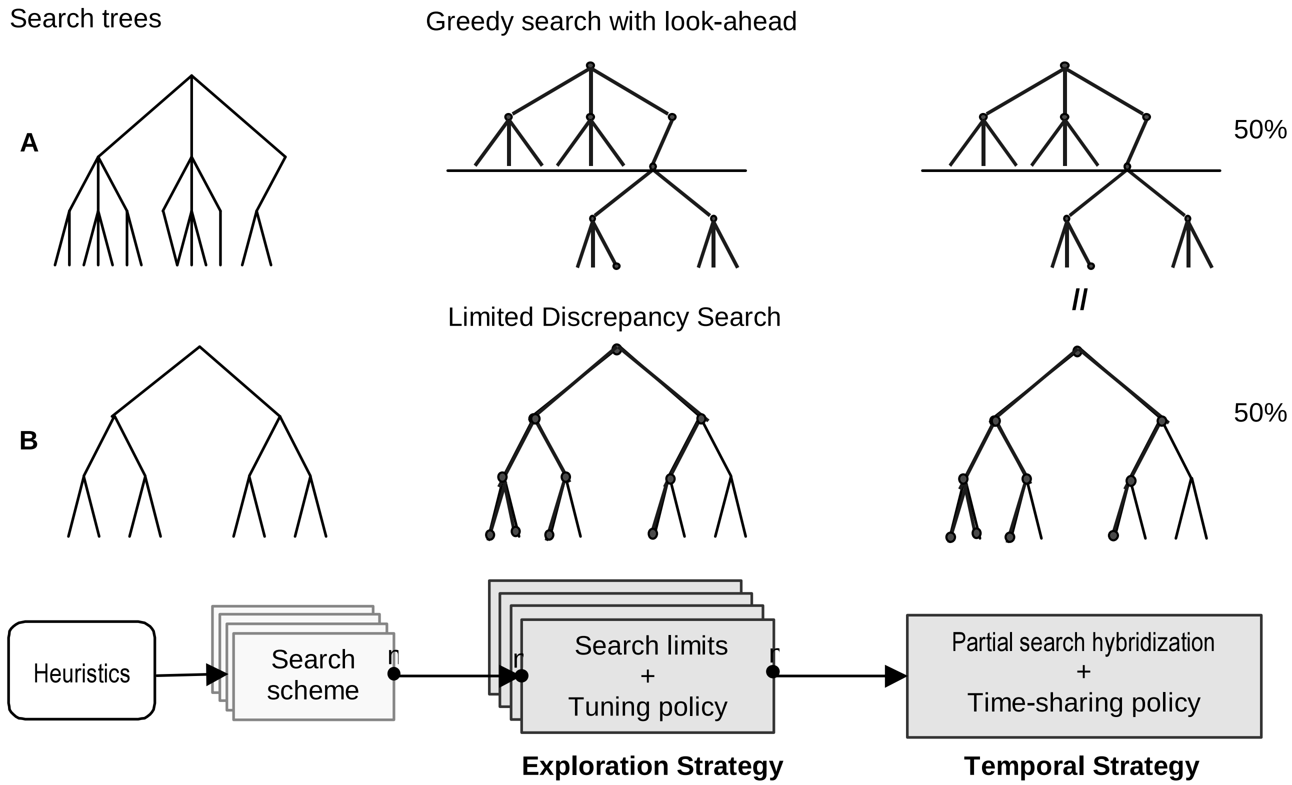

Figure 3: Architecture of a partial search algorithm taking into account a time limit. In the example, the temporal strategy corresponds to the interleaving of two parameterised tree search algorithms applied on two different search schemes (A and B). The time-sharing policy consists in sharing the computational resources equitably.

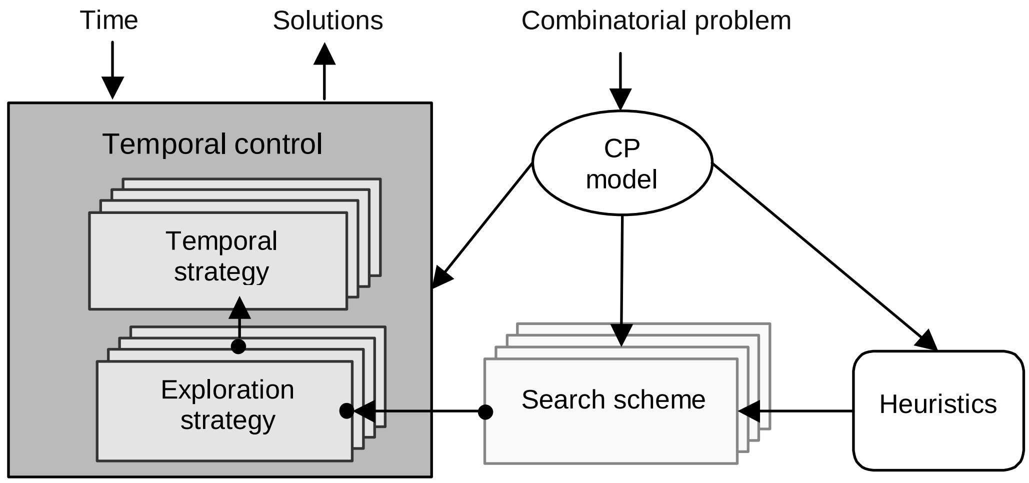

Figure 4: The EOLE architecture derived to the case of partial search.

We are currently investigating how hybrid search algorithms[12], involving both tree search and local search, could be controlled in the same way as we described. The cooperation between searches would be tighter: sharing solutions, cost, value heuristics, and variable heuristics, etc. A blackboard architecture [Corkill 91] can be used to implement the cooperation.

[12] More specifically, the class of cooperative hybridisations, which is described in the next section. Applying a temporal control for the class of intertwined hybridisations is much more complex.

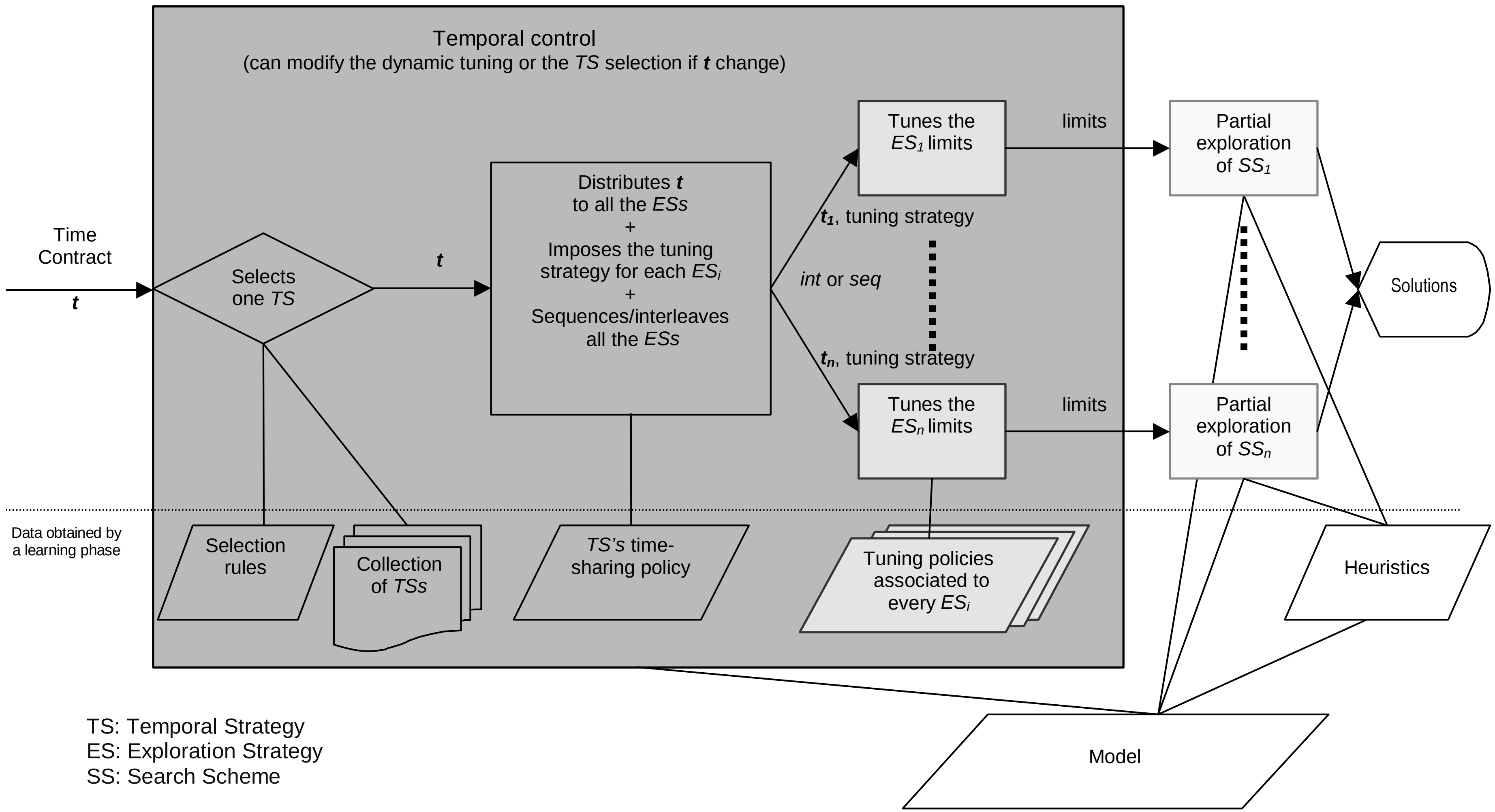

Figure 5: Description of the temporal control of partial search algorithms taking into account a time limit.

4 Hybrid search algorithms

Hybrid algorithms are interesting for on-line optimisation, because they appear as possible compromises between extreme methods. A good hybrid algorithm for on-line optimisation can be parameterised, and its parameters can be tuned such that the realized compromise is adapted to the available computation time (see section 3).

4.1 Components of hybrid algorithms

Hybrid algorithms for combinatorial optimisation borrow their mechanisms from both

- Systematic global search methods, of which the main representative is the Depth First Branch and Bound (DFBB); these methods can be complete.
- Opportunistic local search methods, such as Simulated Annealing and Taboo Search ; these methods are incomplete in general.

A complete method provides guaranteed optimal solutions, but generally within a prohibitive computation time, often not compatible with the needs of on-line optimisation. The main mechanism of global search is refinement, with two facets:

- Decomposition: the ability to decompose instances into unconnected sub-instances,
- Filtering: the ability to locally exclude partial solutions on a sound basis, as being infeasible or sub-optimal.

On the other side, local search is supposed to compute good solutions within a short period of computation time, but offers no guarantee for the quality of the provided solutions. Its main mechanism is the transformation of an instance into a close another one, in a direction guided by some heuristics. Solution repair is another kind of transformation.

4.2 Overview of hybrid algorithms

We present now an inventory of most attractive and relevant existing hybrid methods, in order to point to those most useful for on-line optimisation.

Most proposed hybridisations are intertwined hybridisations: refinements and transformations are closely mingled during computation. Here are the most interesting examples of this kind of hybridisation:

- During a Branch and Bound search, a local search can be done at some choice points, in order to find rapidly good solutions in the corresponding sub-space, and to quickly improve the upper bound (in case of a minimization task). As a special case of this technique, a local search at the root node is often used.
- Shuffling [Applegate & Cook 91], in the field of disjunctive scheduling, is an often-cited technique: a starting complete schedule is partially broken, then rebuilt by mean of a complete algorithm (known as edge-finding), and the process is iterated. [Pesant & Gendreau 99] exploit the same schema in the context of Constraint Programming applied to the TSP (Traveling Salesman Problem).
- [Caseau et al. 99] describe a combination of a limited tree search algorithm (in this case a Limited Discrepancy Search, [Harvey & Ginsberg 95]), improved by constraint filtering. It builds good partial instantiations that are further completed by local search (in this case a Large Neighborhood Search, [Shaw 98]).
- Path-repair [Jussien & Lhomme 00] can be seen as a local search upon the decision sequences leading to a solution.

A different kind of hybridisations, we call cooperative hybridisations, mix different techniques for solving sub-problems or related ones. Here are significant examples:

- [Caseau & Laburthe 95] describe a sequence of different optimisation tasks whose successively: compute lower bounds of the cost of an optimal solution, find a first solution by a greedy algorithm, then launch some local searches (using shuffling, cited above), ending in a global search for a proof of optimality.
- The randomisation and restart technique [Gomez et al. 98] may be seen as a hybrid method. Randomisation roughly consists in making random choices (for variables and values) instead of regular ones, during a tree search. Restart is a special case of transformation, in which a randomised tree search is stopped (after a well chosen amount of time) and then restarted from the beginning. This diversification technique allows for escaping from bad sub-spaces of solutions.
- Many examples may be found in which the main combinatorial problem to be solved can be transformed into a polynomial problem by a slight modification (such as adding or removing some constraint). An optimal solution to the modified problem can be close to an optimal solution of the unmodified problem. In this case, it is often a good idea to first compute an optimal solution of the modified problem and then, starting from this solution, to search for good solutions of the original one by local search. An example of this methodology appears in [Vasquez and Hao 01].

The study of the numerous possible hybridisations of search methods eventually comes out into the following conclusion: the atomic constituents of any search method are:

- Exploration mechanisms: refinements (decomposition, filtering) and transformations (jump in the neighbourhood, repairing),
- Memorization mechanisms, allowing to record still unexplored sub-spaces of solutions,
- Global control search mechanisms.

4.3 A language for hybrid search algorithms

In this paragraph, we describe briefly a simple language the aim of which is to express hybridisations like those sketched above.

Following the works of [Caseau et al. 99] and [Perron 99], the essential idea of this language is to describe hybrid search algorithms by composition of terms. This is illustrated by the following simple example. The term `SEQ(GENERATE(vs1), LOCS(100, vs2))` corresponds to a tree search (`GENERATE`) on the variables of the list vs1, composed sequentially (`SEQ`) with a local search (`LOCS`) on the variables of the list vs2 at each leaf of the search tree (see the figure 2).

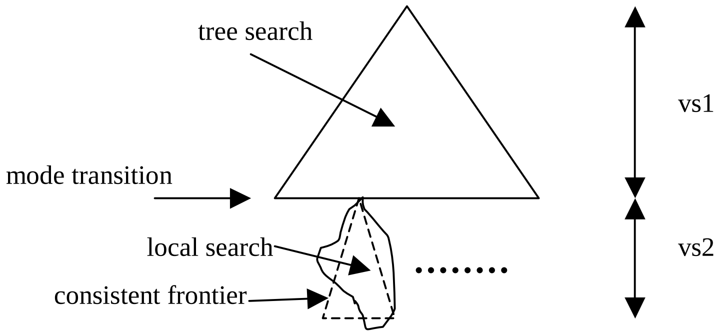

Figure 2: A hybrid search described by the term `SEQ(GENERATE(vs1), LOCS(100, vs2))`. The mode transition corresponds to a switch from a realisable mode (i. e. with propagation) to a relaxed mode (i.e. in which constraints may be violated).

A term denotes a basic search goal or a compound search goal. Basic search goals make up the kernel of the language. Compound search goals are defined from basic search goals, or recursively rewritten from other compound search goals, so this language is extensible.

Here is a significant excerpt of the grammar of the language:

```
Search ::= BasicSearch | CompoundSearch

BasicSearch ::= Decision | BasicComp

Decision ::= Refinement | Transformation

Refinement ::= SETV(Variable, Value) | REMV(Variable, Value) |
                ... | PROP(Integer)

Transformation ::= ASS(Variable, Value) | UNASS(Variable) |
                ... | ENLARGE(Variable, Values)

BasicComp ::= FAIL | SUCCESS | SEQ(Search1, Search2) | ALT(Search1,
Search2)

CompoundSearch ::= GENERATE(Variable) | GENERATE(Variables) | ...
                   | FLIP(Variables) | LOCS(Integer, Variables) | ...
```

There are two kinds of basic search goals: decision goals (`Decision`) and basic composers (`BasicComp`). Decisions can be refinements (basic decisions generally involved in a tree/global search) or transformations (basic decisions generally involved in a local search).

A solver performing hybrid search must be able to reason in (and switch between) two distinct modes: (1) a realisable mode, in which constraints are propagated, and accordingly, domains of variables are reduced, leading

to possible fails; (2) a relaxed mode, in which constraints may be violated without failing, but effects of violations are counted.
Examples of refinement (realisable mode) are:

- SETV, which reduces the domain of the variable to the given value and then propagates this reduction, or fails if the value does not belong to the current domain,
- PROP, which propagates active constraints up to a given level of local consistency.

Examples of transformations (in relaxed mode) are:

- ASS, which assigns a value to a variable, regardless of its current domain,
- UNASS, which restores the initial domain of a variable.

ENLARGE(Variable, Values) is an example of transformation in realisable mode. It restores the set of given values in the domain of the variable, maintaining the consistency of constraints holding on the variable.

SEQ and ALT are basic composers. SEQ(s1, s2) corresponds to conjunction of goals in Prolog [Van Caneghem 86]: all (partial) solutions of the search s1 are starting states for the search s2. ALT(s1, s2) is the disjunction of goals: it performs independently s1 and s2 from the same state.

GENERATE(Variable) and GENERATE(Variables) are compound search terms describing parts of systematic tree search, in the way of OPL [Van Hentenryck 00]. GENERATE(V), where V is a variable, is rewritten as ALT(SETV(V, x), SEQ(REMV(V, x), GENERATE(V))), where x is a value chosen in the current domain of V. In other words, GENERATE(V) sets disjunctive goals, one for each value of the current domain of the variable.
GENERATE(VS), where VS is a list of variables, is roughly rewritten as SEQ(GENERATE(V1), GENERATE(VT)) where V1 is a variable in the list and VT is VS but V1. In other words, GENERATE(VS) describe a search tree performed on the domains of the variables in VS.
On the other hand, FLIP and LOCS are components of local search. FLIP(VS) randomly chooses a variable in the list VS and assigns it a randomly chosen value from its initial domain. LOCS(n, VS) performs a local search involving n flips over variables of VS.
The term LOCS(n, VS) can be rewritten as SEQ(RASS(VS), ITERATE(n, FLIP(VS), PROP(0))) where RASS(VS), initialising the search, is a compound search goal assigning random values to each variable in VS. ITERATE(n, s, f), with n > 0, tries alternatively f (final search goal), and s (iterated search goal) followed by ITERATE(n - 1, s, f):

```
ITERATE(n, s, f) = if (n = 0) FAIL()
else ALT(f, SEQ(s, ITERATE(n - 1, s, f)))
```

Let us give some hints on the foreseen implementation of this language. A hierarchy of classes is built, in bijection with the grammar. The terms are constructors for objects of these classes. As said above, compound search goals are dynamically expanded using rewriting rules, whereas basic terms are processed as direct calls to specialized methods of the solver (CHOCO [Laburthe et al. 00]). As an elementary example, the term SETV gives rise to an object belonging to the SetV class. Objects of this class are processed by the following method (expressed in Claire© [Caseau & Laburthe 96], the implementation language of CHOCO):

```
[solve(p:Problem, s:SetV) : boolean
 -> try (world+(),        // saves the current solver state
          setVal(s.v,s.x), // reduces the domain variable,possibly failing
          propagate(p),    // propagates active constraints,possibly failing
          true)
    catch contradiction   // catches failings
        (world-(),        // restores the solver state (backtrack)
          false)]
```

5 Extensions of constraint solvers to deal with on-line problems

5.1 Local search mechanisms

Constraint solvers are often designed as event schedulers: events corresponding to domain reductions occur and the computation of a consistent state is a sequence of constraint propagation, posting new domain reductions and so on. Local and hybrid search involve non-monotonic moves where domains are not only reduced, but may be changed in any manner. For instance, a flip neighbourhood for the SAT problem amounts to change the value of a variable from 0 to 1 (or conversely): the update of the domain, from {0} to {1} is not a monotonous mode. This section provides a few technical hints on ways to extend the CP framework in order to implement such mechanisms.

The architecture of a CP system can be described by two main components: an event scheduler handling propagation and a global search controller. The scheduler goes through a loop of posting events and waking propagation daemons for propagating constraints. The exploration of the search tree is based on the recursive iteration of the branches associated to a choice point. We here show how both mechanisms can be extended to local search.

For non monotonic moves, propagation needs to be turned off: indeed, since the engine generates infeasible states, constraints may be violated and can therefore no longer be propagated. However, propagation can be replaced by another event-based mechanism where:

- Non monotonic updates are the events. These events may be value flips, domain enlargements or constraint repairs.
- Like monotonic events, each of these events induces a sequence of reactions (maintaining the count of violated constraints or maintaining data structures used for generating the neighbourhood). As suggested in the Localizer [Michel & Van Hentenryck 00] system, such computations can be described by means of invariants, and thus implemented in an event based architecture. The implementation of Localizer has been optimised and pre-compiles the incremental re-computation of events and uses immediate reaction. Other implementations that store events in queues and react to them thereafter are possible. This is for instance the case of the iOpt toolkit [Voudouris et al. 01] that uses a mark-and-sweep algorithm for maintaining invariants. Anyhow, the computations for re-computing a solution from a basic move by invariant maintenance can be described though events and reactions.

For the overall control of the algorithm, the standard scheme consists in changing (improving) a solution with the scope of the moves supported by the neighbourhood. The overall sequence of moves is constructed by applying moves one after the other until some convergence criterion is reached. Although at first sight quite different from search trees, local moves algorithm can be described with the same tools as global search:

- Neighbourhoods can be described by abstract objects whose main method consists in generating a sequence of states: each state is reached by posting an event and processing it. This analogy between branching schemes and neighbourhoods is suggested in Localizer [Michel & Van Hentenryck 00] and developed in Salsa [Laburthe & Caseau 98].
- The overall control of the local search algorithm (multiple descent, taboo, simulated annealing, etc.) can be handled by a controller just like global search algorithms.

Thus, a common architecture based on events, daemons, abstract objects iteratively generating events and controllers can be used for implementing local or global search algorithms. In order to implement hybrids combining both techniques, the engine needs to be able to switch from one mechanism to another. Switching from global search to local search is usually simple and fast: the invariants used by local search are usually simple to initialise. On the contrary, the reverse transition may take longer: all constraints need to be checked and enforced before any reasoning can start. Such a transition from an infeasible instantiation to a feasible one is often done by erasing part of the assignment so that the remainder of the assignment is a consistent partial assignment.

5.2 Conditional propagation on hybrid global constraints

Constraint propagation is the mechanism that controls the interaction of the constraints in order to reduce the search space. Whenever a constraint updates a domain of a variable, the constraint propagation will wake all relevant constraints to detect further consequences. This process is repeated until a fixpoint is reached, i.e. no further inference is possible. Propagation must be combined with search, which is used to assign values to variables.

Possible events on constraints are only domain reduction events (a variable is fixed, the minimum or the maximum has changed). In order to achieve on-line optimisation requirements, there is a need to define new events and mechanisms to control the solving algorithms during the constraint propagation. The conditional propagation is dependent on several parameters like the depth of the search, the remaining time, the type of solution found so far. The aim of this contribution is to define new events enabling the search to drive the algorithmic part of the constraint handling during the constraint propagation.

Constraint handling techniques are a well-known method in CP solvers. Consistency techniques form an important subclass of constraint handling methods. Techniques like arc-consistency and path-consistency are used to remove inconsistent values from the variable domains until solutions are found. Constraint handling techniques in industrial CP system use techniques from Mathematics and Operations Research like techniques from scheduling and placement to check consistency and to remove inconsistent values from the domain of variables.

The new events and mechanisms will allow the user to control applied algorithms at any time of the search during the propagation of the constraints. In consequence, propagation is subject to constraints (time, relaxation, type of algorithms). Our approach provides a simple yet powerful way to drive the algorithmic part of the constraint solver to fulfil the requirements of the on-line optimisation. This facility is based on this basic idea: at any time in the search, a constraint may receive events defining its behaviour when the constraint is awoken during constraint propagation. These events are raised at any time of the search. We distinguish the following events:

- Algorithmic: type of algorithm to apply for constraint handling.
- Relaxation: maximum number of acceptable violated constraints or completely deactivate the constraint.
- Time guarantee: the maximum possible amount of time in handling the constraint.

Efficient algorithms: Efficient CP solvers use efficient and sophisticated algorithms to prune the domain of variable by removing the inconsistent values. The efficiency of such algorithms could be time consuming for on-line optimisation in some situations. Such algorithms are interesting to be applied in the beginning of the search to reduce the number of sub-trees to handle or in some parts of the search to prove quickly the inconsistency.

Verification algorithms: These algorithms are very fast to check elementary constraints or some variants of global constraints. For on-line optimisation, such algorithms are useful especially when the remaining time to handle the problem is getting smaller.

Light algorithms: These algorithms make a compromise between efficient and verification algorithms. Pruning algorithms are applied but expensive ones are inhibited. Some complex algorithms used to handle global constraint are time consuming. For on-line optimisation and in some circumstance they cannot be applied. Meanwhile, there are different degrees of efficiency. A light version of such algorithms is very appropriate for on-line optimisation.

Relaxation: Some global constraints can be seen as a set of elementary constraints. For on-line optimisation, it is not always necessary that all elementary constraints must be satisfied. Partial solutions are acceptable. A global constraint has and additional argument which is either a domain variable or a constant indicating the number of possible violated, relaxed, elementary constraints. In the case of a domain variable, it can be subject to constraint.

Time guarantee

Like the relaxation, at any time of the search, an event can be raised for the constraint to constrain the maximum amount of time of handling the constraint if it is woken during the constraint propagation. For example when a variable becomes ground, such event may cause the awakening of thousands of constraints. In order to ease the implementation of time guarantee, the handling concerns only individual constraint. The time constraint on the total chain of the constraint propagation is controlled by another event that is not discussed in this section. The constraint time is well suited for light and hybrid constraints.

Deactivate constraints

This event enables the user to ignore the handling of some constraints during the constraint propagation. For example, their handling is irrelevant for the current solution.

6 Conclusion

In the EOLE architecture the temporal part is the central one. This component connected with the outside world uses the different sub-components of the system to satisfy incoming requests. It selects and limits the right search strategy. This strategy refers to a particular running definition and can take advantage of particular heuristics. The running strategy uses the model and heuristics. Hence, this selection guided by prediction/learning capacities uses the whole system. Thanks to the temporal module, the search can be adapted to incoming requests. The relaxation/hardening of the temporal constraint can bring new selection/limitation of strategies.

Our current effort is concentrated on the development of the EOLE architecture. The resulting framework will provide:

- A high-level abstraction to help a technical expert to design hybrid search algorithms.
- Time management facilities to control search algorithms with a limit on the computation time.

In order to demonstrate that, the framework will be tested on several real applications in the Telecom domain.

We thank the reviewers for their very helpful comments.

7 Bibliography

[Applegate & Cook 91] D. Applegate, W. Cook. A computational study of the job-shop scheduling problem. ORSA Journal On Computing, 3(2):149--156, 1991.

[Beldiceanu et al. 98] Beldiceanu, N., E. Bourreau, H. Simonis, and D. Rivreau (1998). Introduction de métaheuristiques dans CHIP. *In Proc. of MIC-98*.

[Bresina 96] Bresina, J. L. (1996). Heuristic-Biased Stochastic Sampling. In Proc. of AAAI-96, Portland, OR, pp. 271-278.

[Cabon et al. 98] Cabon, B., S. de Givry, and G. Verfaillie (1998, October 26-30). Anytime Lower Bounds for Constraint Optimization Problems. In Proc. of CP-98, Pisa, Italy, pp. 117-131.

[Caseau & Laburthe 95] Y. Caseau, F. Laburthe. Disjunctive scheduling with task intervals. Technical report, LIENS, École Normale Supérieure de Paris, France, 1995.

[Caseau & Laburthe 96] Caseau, Y. and F. Laburthe (1996). Introduction to the claire programming language. Technical Report LIENS technical report 96-15, Ecole Normale Supérieure.

[Caseau et al. 99] Y. Caseau and F. Laburthe and G. Silverstein. A Meta-Heuristic Factory for Vehicle Routing Problems (Meta-Programming for Meta-Heuristics). In Proc. of CP-99, 144-158, Alexandria, VA, 1999.

[Cheeseman 91] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In J. Mylopoulos and R. Reiter, editors, *Proceedings of IJCAI91*, pages 331-337, San Mateo, CA, 1991. Morgan Kaufmann.

[Chu & Wah 91] Chu, L.-C. and B. W. Wah (1991). Optimization in real time. In Proc. of the Twelfth Real Time Systems Symposium, Washington, D.C., pp. 150-159.

[Cook & Varnell 97] D. J. Cook, R. C Varnell. Maximizing the benefit of parallel search using machine learning. In Proc. AAAI-97, Providence, RI, USA, 1997.

[Corkill 91] D.D. Corkill. Blackboard Systems. *Journal of AI Expert*, 6(9), 40-47, 1991.

[Cormen et al. 94] T. Cormen, C. Leiserson, R. Rivest. Introduction à l'algorithmique. Dunod, 1994.

[de Givry et al. 97] S. de Givry, G. Verfaillie and T. Schiex. Bounding the Optimum of Constraint Optimization Problems. Proc. of CP-97, p. 405-419, Linz, Austria, 1997.

[de Givry 98] de Givry, S. (1998). Algorithmes d'optimisation sous contraintes étudiés dans un cadre temps réel. *Ph. D. thesis*, ENSAE, Toulouse, France.

[de Givry et al. 99] S. de Givry, P. Savéant and Jean Jourdan. Optimisation combinatoire en temps limité: Depth First Branch and Bound adaptatif. *Proceedings of JFPLC99*, p. 161-178, Lyon, 1999.

[Ginsberg & Harvey 92] Ginsberg, M. and W. Harvey (1992). Iterative broadening. Artificial Intelligence 55, 367-383.

[Gomes et al. 98] C. Gomes, B. Selman, H. Kautz. Boosting Combinatorial Search Through Randomization. In Proc. of the 15th National Conference on Artificial Intelligence (AAAI-98), pages 431--437, Madison, WI, USA, 1998.

[Harvey 95] Harvey, W. D. (1995, March). NONSYSTEMATIC BACKTRACKING SEARCH. Ph. D. thesis, Stanford University.

[Harvey & Ginsberg 95] William D. Harvey, Matthew L. Ginsberg. Limited Discrepancy Search. In Proc. of IJCAI-95, pages 607--613, 1995.

[Hogg et al. 96] T. Hogg, B. A. Huberman, C. P. Williams. Phase transitions and the search problem. Artificial Intelligence, 81(1-2):1-15, 1996.

[Jussien & Lhomme 00] N. Jussien, O. Lhomme. Local search with constraint propagation and conflict-based heuristics. In Proc. of the 17th National Conference on Artificial Intelligence (AAAI-00), pages 169--174, Austin, Texas, USA, August 2000.

[Knuth 75] Knuth, D. (1975). Estimating the Efficiency of Backtrack Programs. Mathematics of Computation 29(129), 121-136.

[Korf 90] Korf, R. E. (1990). Real-time heuristic search. Artificial Intelligence 42, 189-211.

[Korf 96] Korf, R. (1996). Improved Limited Discrepancy Search. In Proc. of AAAI-96, Portland, OR, pp. 286-291.

[Laburthe & Caseau 98] F. Laburthe and Y. Caseau (1998, October 26-30). SaLSA: a language for search algorithms. *In Proc. of CP-98*, Pisa, Italy, pp. 310-324.

[Laburthe et al. 00] F. Laburthe and the OCRE project team. CHOCO : implementing a CP kernel. In Proc. of TRICS'2000, Workshop on techniques for implementing Constraint Programming systems, Singapore, 2000.

[Meseguer 97] Meseguer, P. (1997). Interleaved depth-first search. In Proc. of IJCAI-97, Nagoya, Japan, pp. 1382-1387.

[Michel & Van Hentenryck 00] Michel, L. and P. Van Hentenryck (2000). Localizer. *Constraints,* 5(1-2), 43-84.

[Minton 96] S. Minton. Automatically Configuring Constraint Satisfaction Programs: A Case Study. *Constraint*s, 1(1), 1996.

[Langley 92] P. Langley. Systematic and Nonsystematic Search Strategies. In Proc. of AIPS, 145-152, 1992.

[Pemberton & Zhang 96] J. C. Pemberton, X. Zhang. Epsilon-transformation: exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81(1-2):297-325, 1996.

[Perron 99] L. Perron. Search Procedures and Parallelism in Constraint Programming. *In Proc. of CP-99*, 346-360, Alexandria, VA, 1999.

[Pesant & Gendreau 99] G. Pesant, M. Gendreau. A Constraint Programming Framework for Local Search Methods. Journal of Heuristics, 1999.

[Prcovic and Neveu 00] Prcovic, N. and B. Neveu (2000). Recherche à focalisation progressive. In Proc. of JNPC-00, Marseille, France, pp. 191-204.

[Russell & Zilberstein 91] S.J. Russell and S. Zilberstein. Composing Real-Time Systems. Proceedings of IJCAI-91, p. 212-217, Menlo Park, CA, 1991.

[Selman et al. 92] B. Selman and H. Levesque and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. Proceedings of AAAI-92, p. 440-446, San Jose, CA, 1992.

[Shaw 98] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. Proc. of CP-98, LNCS 1520, 1998.

[Van Caneghem 86] M. Van Caneghem. L'Anatomie de Prolog. *InterEditions*, Paris, 1986.

[Van Hentenryck 00] P. Van Hentenryck. ILOG OPL 3.0, Optimization Programming Language, Reference Manual. ILOG, Janvier 2000.

[Vasquez & Hao 01] M. Vasquez, J.-K. Hao. A Hybrid Approach for the 0-1 Multidimensional Knapsack Problem. *In Proc. of IJCAI-01*, 328-333, 2001.

[Voudouris et al. 01] C. Voudouris, R. Dorne, D. Lesaint and A. Liret. iOpt: A Software Toolkit for Heuristic Search Methods. In Proc. of CP-01, Paphos, Cyprus, 2001.

[Walsh 97] T. Walsh. Depth-bounded discrepancy search. *In Proc. of IJCAI-97*, Nagoya, Japan, 1997.

[Zilberstein 96] S. Zilberstein. Using Anytime Algorithms in Intelligent Systems. AI Magazine, 17(3):73-83, 1996.

[Zhang, 1998] Zhang, W. (1998). Complete anytime beam search. In Proc. of AAAI-98, Madison, WI.