Search: from Algorithms to Systems

Youssef Hamadi
Microsoft Research,
7 J J Thomson Avenue,
Cambridge CB3 0FB, United Kingdom

LIX École Polytechnique,
F-91128 Palaiseau, France
youssefh@microsoft.com

July 13, 2012

Contents

List of Figures

List of Tables

Chapter 1

Acknowledgments

Chapter 2

Introduction

This document summarizes some of my post PhD contributions. They were carried out while working at Microsoft Research, one of the largest software research organizations in the world. This unique context gave me the chance to be exposed to various research problems and application domains. This often led to fruitful collaborations in areas where some expertise in combinatorial search is beneficial. This mainly includes work on Reconciliation in distributed database [CH06, HS05, SBCH04], Scheduling for large scale Data Centers [Ham05, BFH+05], application to Digital Photography [RBHB06], Business Intelligence [HQ06], Computational Biology [CBH+07], Knowledge Compilation [SBH07], Sequential Satisfiability [PHS09, PHS08, ABH+08], new SMT theories for improved Software Verification [WHdM10], and Computational Sustainability [FHS12, FHS11]. None of these works will be exposed here.

Instead, I will focus my presentation on contributions to general combinatorial search for the constraint satisfaction (CSP), and propositional satisfiability (SAT) formalisms. I will develop several advanced techniques which bring search algorithms closer to fully fledged complex systems capable of self performances analysis to correct and improve their search strategies.

This document contains two parts. The second part presents a curriculum vitae which provides along an extensive publication list, an overview of my professional engagements including student supervisions, and teaching activities. In the first part, I present a synthesis of my research. The presentation is gradual, and describes search systems of raising complexity and efficiency.

The Chapter 3 presents portfolios of distributed CSP algorithms which demonstrate that competition and cooperation can improve the performance of existing distributed search techniques by several orders of magnitude. We show that a portfolio approach makes a better use of computational resources by reducing the idle time of agents. Indeed, by allowing search agents to simultaneously work at different tree-search levels it provides a solution to the classical work imbalance problem of distributed backtracking. It also shows the value of knowledge sharing to significantly speed-up search and provide portfolios whose performance is better than any constituent algorithm.

The previous notions are then applied to the important problem of propositional satisfiability in Chapter 4. Parallel SAT became important in the last few years from the practical solving successes of modern SAT solvers combined to the wide availability of multicore computing platforms. We present ManySAT, the first parallel SAT portfolio which completely redefined the parallel SAT landscape. In ManySAT different modern SAT solvers are organized around a cooperative framework to quickly solve a given instance. ManySAT brings portfolios one step closer to fully fledged search systems with the addition of control theory techniques to incrementally adjust the relevance of the knowledge shared.

The Chapter 5 considers Constraint Programming, and presents Continuous

Search (CS). In CS, we move to complex solving systems which incrementally learn a predictive model able to accurately match instances features to good solver's parameters. CS interleaves two functioning modes. In exploitation mode, the instance submitted by the user is processed by the constraint solver; the current heuristics model is used to parameterize the solver depending on the instance at hand. In learning or exploration mode, CS reuses the last submitted instance, running other heuristics than the one used in exploitation mode in order to find which strategy would have been most efficient for this instance. New information is thus generated and exploited in order to refine the heuristics model, in a transparent manner: without requiring the user's input and by only using the idle computer's CPU cycles. CS acts like an autonomous search system able to analyses its performances and gradually correct its search strategies.

To conclude the presentation of our research, we present the concept of Autonomous Search in Chapter 6. We define autonomous solvers as solvers that contain control in their search process, and study such autonomous systems w.r.t. their specific control methods. A control process includes a strategy that manages the modification of some of the solver's components and behavioral features after the application of some solving functions. The overall strategy to combine and use components and parameters can be based on learning that uses information from the current solving process and/or from previously solved instances. This chapter proposes a taxonomy of search processes w.r.t. their computation characteristics, and provides a rule-based characterization of autonomous solvers. This allows a formalizing of solvers adaptations and modifications with computation rules that describe the modification of the solver's components transformation.

Part I

Research Synthesis

Chapter 3

Distributed Constraint Networks

The work presented in this Chapter finds its premises in the perspectives of my PhD manuscript [Ham99b]. It was put to test during the summer internship of Georg Ringwelski at MSR Cambridge. It led to two publications and to one US patent:

- Georg Ringwelski and Youssef Hamadi. Boosting distributed constraint satisfaction. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 549–562. Springer, 2005.
- Youssef Hamadi and Georg Ringwelski. Boosting distributed constraint satisfaction. *J. Heuristics*, 17(3):251–279, 2011.
- US Patent No. 7472094, Multi-Ordered Distributed Constraint Search, Youssef Hamadi, and Georg Ringwelski, Dec 30, 2008 (filed in 2005).

3.1 Introduction

In combinatorial tree-based search, finding a good labelling strategy is a difficult and tedious task which usually requires long and expensive preliminary experiments on a set of representative problem instances. Performing those experiments or defining realistic input samples is far from being simple for today's large scale real life applications. The previous observations are exacerbated in the processing of distributed constraint satisfaction problems (DisCSPs). Indeed, the distributed nature of those problems makes any preliminary experimental step difficult since constrained problems usually emerge from the interaction of independent and disconnected agents transiently agreeing to look after a set of globally consistent local solutions [FM02].
This work targets on those cases where bad performance in DisCSP can be prevented by choosing a good labelling strategy i.e., decide on an ordered set of variables and values pairs to branch on, and executing it in a benefiting order within the agents. In the following, we define a notion for the risks we have to face when choosing a strategy and present the new Multi-directional Search Framework or M-framework for the execution of distributed search. An M- portfolio executes several distributed search strategies in parallel and lets them compete for being the first to finish. Additionally, cooperation of the distributed searches is implemented with the aggregation of knowledge within agents. The knowledge gained from *all* the parallel searches is used by the agents for their local decision making in each single search. We present two principles of aggregation and employ them in communication-free methods, applicable to the limited scope of the agents in DisCSP.
Each DisCSP agent still has access to only a subset of the variables as usual but itself runs several copies of the search process on these variables under different "search contexts", potentially integrating information across these different contexts. Since these contexts have different indirect information about other agents (based on the messages they have received), this indirectly allows aggregating information across different agents as well.

We apply our framework in two case studies where we define the algorithms M-ABT and M-IDIBT that improve their counterparts ABT [YDIK92] and IDIBT [Ham02b] by several orders of magnitude. With these case studies we can show the benefit of competition and cooperation for the underlying distributed search algorithms. We expect the M- framework to be similarly beneficial for other tree-based DisCSP algorithms. The framework presented here may be applied to them in a straightforward way that is described in this chapter.

3.2 Related Work

The benefit of cooperating searches executed in parallel was first investigated for CSP in [HH93]. They used multiple agents, each of which executed one monolithic search algorithm. Agents cooperated by writing/reading hints to/from a common blackboard. The hints were partial solutions or nogoods its sender has found and the receiver could re-use them in its efforts. In contrast to our work, this multi-agent system was an artifact created for the cooperation. Thus the overhead it produced, especially when not every agent could use its own processor, added directly to the overall performance. Another big difference between Hogg's work and ours is that DisCSP agents do not have a global view of the searches and can thus only communicate what's in their agent-view which usually captures partial solutions for comparably few variables only.

Later the expected performance and the expected (Randomization-) risk in portfolios of algorithms was investigated in [GS97, GS01]. No cooperation between the processes was used here. In the newer paper the authors concluded that portfolios, provided there are enough processors, reduce the risk and improve the performance. When algorithms do not run in parallel (i.e., when it is not the case that each search can use its own processor) the portfolio approach becomes equivalent to random restarts [GSK98]. Using only one processor, the expected performance and risk of both are equivalent. In contrast to Gomes and Selman we cannot allocate search processes to CPUs. In DisCSP we have to allocate each agent, that participates in every search, to one process. Consequently parallelism is in our setting not an overhead-prune artifact. We distribute our computations to the concurrent processes. However, this is done in a different way than in [GS01], we do not assign each search to one process, but each search is temporarily performed in each process. Or from the other perspective, each agent participates in all the concurrent search efforts at the same time. Thus load-balancing is performed by the agents and not by the designer of the portfolio. In this work we consider agents that do this on a first-come-first-serve basis. Another major difference to Gomes' and Selman's work is that we use cooperation (Aggregation) between the agents.

Recent work on constraint optimization [CB04] has shown that letting multiple search algorithms compete and cooperate can be very beneficial without having to know much about the algorithms themselves. They successfully use various optimization methods on one processor which compete for finding the next best solutions. Furthermore they cooperate by interchanging the best known feasi-

ble solutions. However, this method of cooperation cannot be applied to our distributed constraint satisfaction settings for two reasons: first, we do not have (or want) a global view to a current variable assignment and second we have no reliable metric to evaluate partial assignments in CSP.
Concurrent search in DisCSP [ZM05, Ham02b, Ham02a] differs from M- in a significant way. These approaches also use multiple contexts in parallel to accelerate search. However, in the named work certain portions of the search space are assigned to search efforts. These work apply divide-and-conquer approaches. In the framework presented here we do not split the search space but let every context work on the complete problem. This makes a significant difference in the application of both concepts; M- is a framework while divide-and-conquer is a class of algorithms. M- requires algorithms to do the work while making use of available resources to try multiple things in parallel. Consequently concurrent search could be integrated in M- by letting multiple concurrent search-algorithms (each hosting multiple concurrent searches) run in parallel.
In DisCSP research many ways to improve the performance of search have been found in recent years, including for example [YD98, BBMM05, ZM05, SF05, MSTY05]. All of the named approaches can be integrated easily in the M- framework. The steps to take in order to do this were described in this chapter. The data structures have to be generalized to handle M contexts, and the search functions and procedures have to integrate an extra *context* parameter during their execution. Depending on the algorithm we may achieve heterogeneous portfolios in different ways. In this work we demonstrated the use of different agent topologies but other properties of algorithms can similarly be diversified in a portfolio. As described in the previous paragraph, the main difference between the work presented here and the named DisCSP research is that we do not provide but require a DisCSP algorithm to serve as input to create an instance of M-.
A different research trend performs "algorithm selection" [Ric76]. Here, portfolio does not represent competing methods but complementary ones. The problem is then to select from the portfolio the best possible method in order to tackle some incoming instance. [XHHLB07, LBNA$^+$03] applies the previous to combinatorial optimization. The authors use portfolios which combine algorithms with uncorrelated easy inputs. Their approach requires an extensive experimental step. It starts with the identification of problem's features which are representative of runtime performances. These features are used to generate a large set of problem instances which allow the collection of runtime data for each individual algorithm. Finally, statistical regression is used to learn a real-valued function of the features which allows runtime prediction. In real situation, the previous function predicts each algorithm's running time and the real instance is solved with the algorithm identified as the fastest one. The key point is to combine uncorrelated methods in order to exploit their relative strengths. The most important drawback here is the extensive offline step. This step must be performed for each new domain space. Moreover a careful analysis of the problem must be performed by the end-user to identify key parameters. The

previous makes this approach highly unrealistic in a truly distributed system made by opportunistically connected components [FM02]. Finally knowledge sharing is not applicable in this approach.

3.3 Background

In this section we define some notions used later in the chapter. We briefly define the problem class considered, two algorithms to solve them and three metrics to evaluate the performance of these algorithms.

3.3.1 Distributed Constraint Satisfaction Problems

DisCSP is a problem solving paradigm usually deployed in multi-agent applications where the global outcome depends on the joint decisions of autonomous agents. Examples of such applications are distributed planning [AD97], and distributed sensor networks management [FM02]. Informally, a DisCSP is represented by a set of variables, each of which is associated with a domain of values, and a set of constraints that restrict combinations of values between variables. The variables are partitioned amongst a set of agents, such that each agent owns a proper subset of the variables. The task is for each agent to assign a value to each variable it owns without violating the constraints.
Modelling a distributed problem in this paradigm involves the definition of the right decision variables (e.g., in [FM02] one variable to encode the orientation of the radar beam of some sensor) with the right set of constraints (e.g., in [FM02] at least three sensors must agree on the orientation of their beams to correctly track a target).
Solving a DisCSP is equivalent to finding an assignment of values to variables such that all the constraints are satisfied.
Formally, a DisCSP is a quadruplet (X,D,C,A) where:

1. X is a set of n variables $X_1, X_2, \ldots, X_n$.

2. D is a set of domain $D_1, D_2, \ldots, D_n$ of possible values for the variables $X_1, X_2, \ldots, X_n$ respectively.

3. C is a set of constraints on the values of the variables. The constraint $C_k\left(X_{k1}, \ldots, X_{kj}\right)$ is a predicate defined on the Cartesian product $D_{k1} \times \ldots \times D_{kj}$. The predicate is true if the value assignment of these variables satisfies the constraint.

4. $\mathsf{A} = \{A_1, A_2, \ldots, A_p\}$ is a partitioning of X amongst p autonomous processes or agents where each agent A_k "owns" a subset of the variables in X with respect to some mapping function $f : \mathsf{X} \rightarrow \mathsf{A}, s.t.\ f(X_i) = A_j$.

A basic method for finding a global solution uses the distributed backtracking paradigm [YDIK92]. The agents are prioritized into a partial ordering graph

such that any two agents are connected if there is at least one constraint between them. The ordering is determined by user-defined heuristics. Solution synthesis begins with agents finding solutions to their respective problems. The local solutions are then propagated to respective children i.e., agents with lower priorities. This propagation of local solutions from parent to child proceeds until a child agent is unable to find a local solution. At that point, a *nogood* is discovered. These elements record inconsistent combinations of values between local solutions, and can be represented as new constraints. Backtracking is then performed to some parent agent and the search proceeds from there i.e., propagation of an alternative local solution or new backtrack. The detection and the recording of inconsistent states are the main features which distinguish distributed backtracking algorithms. This process carries on until either a solution is found or all the different combinations of local solutions have been tried and none of them can satisfy all the constraints. Since these algorithms run without any global management point, successful states - where each agent has a satisfiable local solution - must be detected through some additional termination detection protocol (e.g., [CL85]).

3.3.2 DisCSP Algorithms

As a case study to investigate the benefit of competition and cooperation in distributed search we applied our framework to the distributed tree-based search IDIBT [Ham02b] and ABT [YDIK92].

IDIBT exploits the asynchronous nature of the agents in a DisCSP to perform parallel backtracking. This is achieved by splitting the solution space of the top priority agent into independent sub-spaces. Each sub-space combined with the remaining parts of the problem represents a new sub-problem or context. In each context, the same agent ordering is used. Globally, the search is truly parallel since two agents can simultaneously act in different sub-spaces. At the agent level, search contexts are interleaved and explored sequentially.

This divide and conquer strategy allows the algorithm to perform well when the value selection strategy is poorly informed. Beside this parallelization of the exploration, IDIBT uses a constructive approach to thoroughly explore the space by an accurate book-keeping of the explored states. It does not add nogoods to the problem definition. However, it often requires the extension of the parent-child relation to enforce the completeness of the exploration.

In this work, IDIBT agents use exactly one context to implement (each) distributed backtracking. Please note that we also use contexts but in a different way. We only use them to implement our portfolio of variable-orderings. In contrast to [Ham02b] we thus apply each of them to the complete search tree.

IDIBT requires a hierarchical ordering among the agents. Agents with higher priority will send their local solution through infoVal-messages to agents with lower priority. In order to set up a static hierarchy among agents, IDIBT uses the DisAO algorithm [Ham02b]. In this chapter we do not use DisAO but define an order a priori by hand. However, the DisAO has an extra functionality which is essential for the correctness of IDIBT: it establishes extra links between agents

which are necessary to ensure that every relevant backtrack-message is actually received by the right agent. In order to prevent this pre-processing of the agent topology with DisAO we changed the IDIBT algorithm to add the required extra links between agents dynamically during search (similar to the processing of "addLink"-messages in ABT). Finally we extended the algorithm to support dynamic value selection which is essential for the Aggregation described later in this chapter.

ABT is the most prominent tree-based distributed search algorithm. Just as IDIBT it uses a hierarchy to identify the receivers of messages that inform others of currently made choices, of the need to backtrack or the need to establish an extra link. In contrast to IDIBT, ABT uses a nogood-store to ensure completeness.

In this work, we used ABT in its original version where the hierarchy of agents is given a priori.

Remark that even if IDIBT is used with a single context in our experiments, that does not make it similar to ABT. Indeed, IDIBT does not record nogood, while ABT does. This make a huge difference between these algorithms.

3.3.3 Performance of DisCSP Algorithms

The performance of distributed algorithms is comparably hard to capture in a meaningful way. The challenge is to find a metric which includes the complexity of the locally executed computations and the need for communication while reducing this by the work that can practically be done in parallel. The community has proposed different metrics which meet these requirements.

Non Concurrent Constraint Checks Constraint checks (cc) is an established metric to express the effort of CSP algorithms. It is the number of queries made to constraints whether they are satisfied with a set of values or not. Non Concurrent Constraint Checks (nccc) [GZG+08] apply this metric to a concurrent context. nccc counts the constraint checks which *cannot* be made concurrently. When two agents A and B receive information about a new value from another agent C, they then can check their local consistency independently and thus concurrently. Assuming this costs 10 constraint checks each, it will be 20 cc but only 10 nccc. However, when agent C needs 10 cc to find this value this is not independent of A and B and will result in 20 nccc and 30 cc respectively.

Sequential Messages Counting messages (mc) is an established method to evaluate the performance of distributed systems. The number of messages is relevant because their transportation often requires much more time than local computations. Analogously to counting cc in distributed systems we also have to distinguish the messages that can be sent concurrently [Lam78]. This also applies to DisCSP [SSHF00]. If an agent C informs two agents A and B of its new value then it uses two messages. However, the 2 mc will only count as one sequential message (smc) because both are independent and can be sent in parallel. When agent A now replies to this message then we will have two smc (and 3 mc), because the reply is dependent of the message sent by C. The metric

thus refers to the longest sequence of messages that is sent for the execution of the algorithm.
Parallel runtime Runtime is a popular metric in practice today. It expresses in a clear and easily understandable way the actual performance of an algorithm. Its drawback is that it is hardly comparable when using different hardware. In multi-tasking operating systems we usually use CPU time in order to capture just the time the considered process requires. Again, in concurrent systems this metric cannot be applied so easily. We have multiple processes and CPUs which share the workload. In order to capture parallel runtime (pt) we have to track dependencies of computations and accumulate the dependent runtime required by different processes. The longest path through such depending activities will be the required parallel time. In simulators of distributed systems which run on one processor we can capture the pt in the same way. With every message we transmit the pt required so far. The receiver will add the time it needs to process the message and pass the sum on with the next (depending) message.

3.4 Risks in Search

Here we present two definitions of *risk* in search. Both kinds of risks motivate our work. We want to reduce the risk of poor performance in DisCSP. The first notion called *randomization risk* is related to the changes in performances when the same non-deterministic algorithm is applied multiple times to a single problem instance. The second notion called *selection risk* represents the risk of selecting the wrong algorithm or labeling strategy, i.e., one that performs poorly on the considered problem instance.

3.4.1 Randomization Risk

In [GS01] "risk" is defined as the standard deviation of the performance of one algorithm applied to one problem multiple times. This risk increases when more randomness is used in the algorithms. With random value selection for example it is high and with a completely deterministic algorithm it will be close to zero. In order to prevent confusion we will refer to this risk as the Randomization-Risk (R-Risk) in the rest of the chapter.

Definition 1. *The R-Risk is the standard deviation of the performance of one algorithm applied multiple time to one problem.*

In asynchronous and distributed systems we are not able to eliminate randomness completely. Besides explicitly intended randomness (e.g., in value selection functions) it emerges from external factors including the CPU scheduling of agents or unpredictable times for message passing [ZM03].
Reducing the R-Risk leads in many cases to trade-offs in performance [GSK98], such that the reduction of this risk is in general not desirable. For instance, we would in most cases rather wait between 1–10 seconds for a solution than

waiting 7–8 seconds. In the latter case the risk is lower but we do not have the chance to get the best performance.

Moreover, increasing randomization and thus the R-risk is known to reduce the phenomenon of heavy-tail behaviour in search [Gom03]. Heavy-tailedness exposes the phenomena that wrong decisions made early during search may lead to extensive thrashing and thus unacceptable performance. In a preliminary experiment we could detect this phenomenon in DisCSP with the algorithms ABT and IDIBT. We used lexicographic variable and value selection to solve 20 different quasigroup completion problems [GW]. A quasigroup is an algebraic structure resembling a group in the sense that "division" is always possible. Quasigroups differ from groups mainly in that they need not be associative.

The problems were encoded in a straightforward model: N^2 variables, one variable per agent, no symmetry breaking, binary constraints only. We solved problems with a 42% ratio of pre-assigned values which is the peak value in the phase transition for all orders, i.e., we used the hardest problem instances for our test. Each problem was solved 20 times resulting in a sample size of 400. With ABT we solved problems of order 6 and with the faster IDIBT problems of order 7. Randomness resulted from random message delays and the inpredictable agent activation of the simulator.

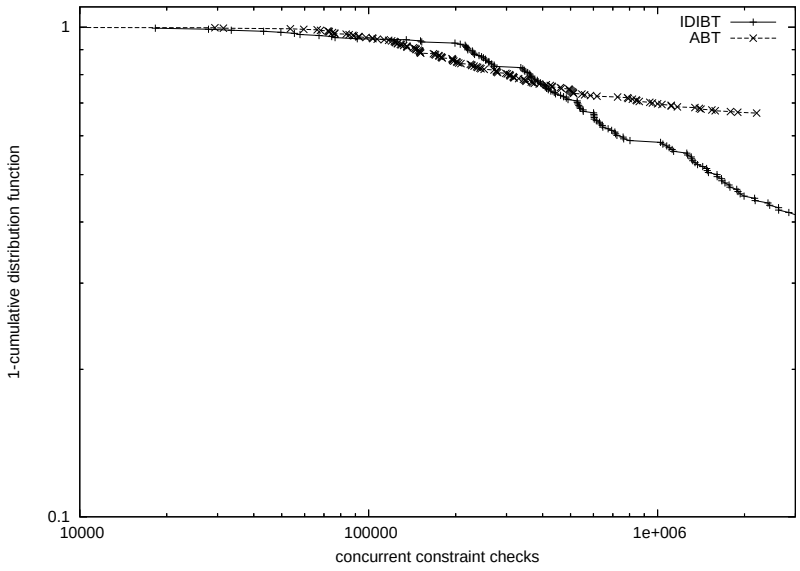


Figure 3.1: Heavy-tail behaviour of IDIBT and ABT.

The results of this experiment are presented in Figure 3.1. We can observe a linear decay of the cumulative distribution function of ABT on a log-log scale. For IDIBT, since this algorithm is more performant than ABT, the linear decay is not visible, but would have been apparent at a different scale, i.e., for the processing of larger problems. The cumulative distribution function of x gives us the probability (y-axis) that the algorithm will perform worse than x. It can be seen that the curves display a pareto-distribution having a less than exponential decay. A pareto-distribution or power law probability distribution is seen in many natural phenomenon (wealth distribution, sizes of sand particles, etc.) it implies that the phenomenon under consideration distribute a particular characteristic in an unbalanced way, e.g., "80-20 rule" which says that 20% of the population controls 80% of the wealth.
This hyperbolic (i.e., less than exponential) decay is identified on the log-log scale when the curves look linear. This is a common means of characterizing a heavy tail [Hil75]. Thus, we could (for the first time) observe heavy tails for both considered DisCSP algorithms in these experiments.
In order to diminish the heavy tail Gomes and Selman propose the use of random restarts during search. With this technique we interrupt thrashing and restart search once the effort does not seem promising anymore. Nowadays, restart is an essential part of any modern tree-based SAT solver [BHZ06], and is also successfully applied to large scale CP applications [OGD06].
With a central control this decision to restart can be based on information gained from a global view on the search space e.g., overall number of fails or backtrack decisions. In DisCSP we do not have such a global view and could thus only decide locally either to restart or to keep trying. However, the local view may not be informed enough for this decision. In these algorithms different efforts are concurrently made on separate sets of variables. Thus we must face the risk that while one effort may thrash and identify the need to restart, another effort may have almost solved its sub-problem. Furthermore, stopping and restarting a distributed system is costly since it involves extra communications. It requires a wave of messages to tell all agents to stop. After that, global quiescence has to be detected before a restart can be launched. Thus, we do not consider restarts to be very practical for DisCSP.
In [GS01] the authors incorporate random restarts in a different way. When we use a portfolio of algorithms performing random searches in parallel then this can be equivalent to starting all of these algorithms one after each other in a restart-setting. They showed that, if one processor is available, the use of portfolios of algorithms or labeling strategies has equivalent performance as the application of random restarts. When we use a portfolio of random searches in parallel sharing the same computational resources then the expected value of the performance is the same as running these random searches one after each other using random restarts. If we have more than one processor, the performance may increase.
In this chapter we make use of this in order to reduce heavy-tail behaviour in DisCSP. We use portfolios as a surrogate of random restarts to reduce the risk of extensive thrashing paralyzing the algorithm. This will reduce the risk

of very slow runs and thus reduce the R-risk as well, and improve the mean runtime. The randomness may result from random value selection or from the distribution itself (message transportation and process activation). As we will show in Section 3.6 we can avoid heavy-tailedness with this new technique.

3.4.2 Selection Risk

The risk we take when we select a certain algorithm or a heuristic to be applied within an algorithm to solve a problem will always be that this is the wrong choice. For most problems we do not know in advance, which algorithm or heuristic will be the best and may select one which performs much worse than others. We'll refer to this risk as to the Selection-Risk (S-Risk).

Definition 2. *The S-Risk of a set of algorithms/heuristics A is the standard deviation of the performance of each $a \in A$ applied the same number of times to one problem.*

We investigated the S-Risk emerging from the chosen agent ordering in IDIBT in a preliminary experiment on small, fairly hard random problems (15 variables, 5 values, density 0.3, tightness 0.4). These problems represent randomly generated CSPs where the link density between variables is set to 30%, whereas the tightness density of each constraint is set to 40%, i.e., 40% of the values combinations are disabled in each constraint. We used one variable per agent and could thus implement variable-orderings in the ordering of agents. We used lexicographic value selection and four different static variable-ordering heuristics: a well-known "intelligent" heuristic (namely maxDegree), its inverse (which should be bad) and two different blind heuristics. As expected, we could observe that the intelligent heuristic dominates in average but that it is not always the best. It was the fastest in 59% of the tests, but it was also the slowest in 5% of the experiments. The second best heuristic (best in 18%) was also the second worst (also 18%). The "anti-intelligent" heuristic turned out to be the best of the four in 7% after all. The differences between the performances were quite significant with a factor of up to 5. Applied to the same problems, ABT gave very similar results with a larger performance range of up to factor 40.

3.5 Boosting Distributed Constraint Satisfaction

In DisCSP the variable ordering is partially implied by the agent topology. Neighboring agents will have to be labelled directly after each other. For example, if each agent hosts one variable then for each constraint a connection between two agents/variables must be imposed. From this follows that the thus connected variables are labelled directly one after the other because they communicate along this established link. In other topologies where we have inner and outer constraints, naturally only the outer constraints must be implemented as links between agents and we have free choice of variable selection inside the nodes.

For the inter-agent constraints we have to define a direction for each link. This direction defines the priority of the agents [YDIK92] and thus the direction in which backtracking is performed. It can be chosen in any way for each of the existing connections. In Figure 3.2 we show two different static agent-topologies emerging from two different variable-ordering heuristics in DisCSP.

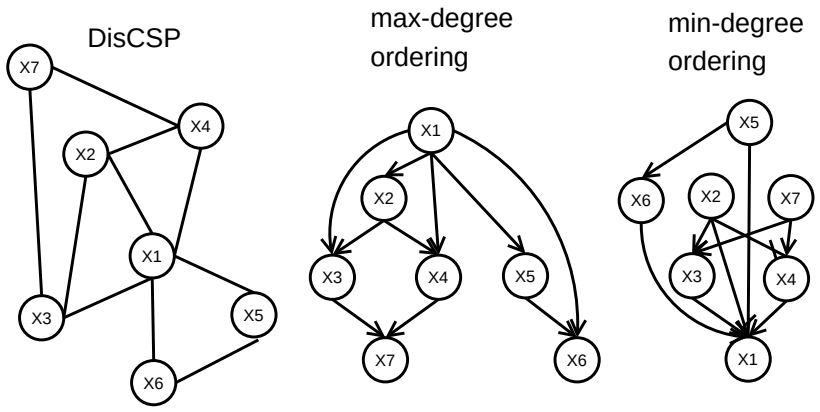


Figure 3.2: DisCSP (left) and agent topologies implied by the variable orderings max-degree (middle) and min-degree (right).

3.5.1 Utilizing Competition With Portfolios

The idea presented in this chapter is that several variable orderings and thus several agent topologies are used by concurrent distributed searches. We refer to this idea as the M-framework for DisCSP. Applied to an algorithm X it defines a DisCSP algorithm M-X which applies X multiple times in parallel. Each search operates in its usual way on one of the previously selected topologies. In each agent the multiple searches use separate contexts to store the various pieces of information they require. These include for example adjacent agents, their current value, their beliefs about the current values of other agents, etc.
In Figure 3.3 we show how an agent hosting variable X_4 from Figure 3.2 could employ the two described variable orderings. The figures shows the internal information, and the associated pseudo code. On the right part of the figure, we can see that this agent hosts two different current values, one for each search and two different agent-views which contain its beliefs about the values of higher-priority agents. The set of these higher-priority agents depends on the chosen topology and thus on the chosen variable ordering. The figure also shows on the left the pseudo-code associated with some tree-based search algorithm. There, the function and procedures are augmented with an extra *context* parameter, which is used to access the right subset of data.
In a M- search, *different* search-efforts can be made in parallel. Each message

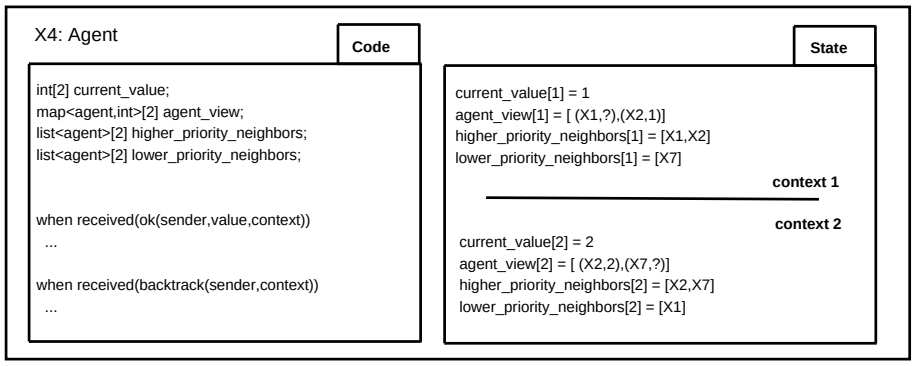


Figure 3.3: Two contexts for the agent hosting X_4 from Figure 3.2 resulting from two variable orderings.

will refer to a context and will be processed in the scope of this context. The first search to terminate will deliver the solution or report failure. Termination detection has thus to be implemented for each of the contexts separately. This does not result in any extra communication as shown for the multiple contexts of IDIBT in [Ham02b].
With the use of multiple contexts we implement a portfolio of heuristics which is known to reduce the heavy-tail of CSP [GS01]. As we will show in our experiments this is also beneficial for DisCSP. In contrast to random restarts we do not stop any search although it may be stuck due to bad early choices. We rather let such efforts run while concurrent efforts may find a solution. As soon as a solution is detected in one of the contexts all searches are stopped.
Additionally, we can reduce the S-Risk by adding more diversity to the used portfolio. Assuming we do not know anything about the quality of orderings, the chance of including a good ordering in a set of M different orderings is M-times higher than selecting it for execution in one search. When we know intelligent heuristics we should include them but the use of many of them will reduce the risk of bad performance for every single problem instance (cf. experiment in Section 3.4.2). Furthermore, the expected performance is improved with the M-framework since always the best heuristic in the portfolio will deliver the solution or report failure. If we have a portfolio of orderings M where the expected runtime of each $m \in M$ is $t(m)$ then ideally (if no overhead emerges) the system terminates after $min(\{t(m)|m \in M\})$.

3.5.2 Utilizing Cooperation With Aggregation

Besides the idea of letting randomized algorithms compete such that overall we are always "as good as the best heuristic" the M-framework can also use cooperation. Cooperation through knowledge-sharing is a very powerful concept which allows a collection of agents to perform even better than the best of them.

As suggested by Reid Smith, $Power = Knowledge^{Shared}$, where the exponent represents the number of agents whose knowledge is brought to the problem [Buc06]. With this, M- portfolios may be able to accelerate the search effort even more by providing it with useful knowledge others have found. Cooperation is implemented in the aggregation of knowledge *within* the agents. The agents use the information gained from one search context to make better decisions (value selection) in another search context. This enlarges the amount of knowledge on the basis of which local decisions are made.
In distributed search, the only information that agents can use for aggregation is their view of the global system. With multiple contexts, the agents have multiple - sometime orthogonal - views and thus more information available for their local reasoning. Since all these views are recorded by each individual agent within his local knowledge base, sharing inter-context information is costless. It is just a matter of reading in the local knowledge base what has been decided for context c, in order to make a new decision in context c'. In this setting, the **aggregation yields no extra communication costs** (i.e., no message passing). It is performed locally and does not require any messages or accesses to some shared blackboard.

3.5.3 Categories of Knowledge

In order to implement Aggregation we have to make two design decisions: first, which knowledge is used and second, how it is used. As mentioned before we use knowledge that is available for free from the internally stored data of the agents. In particular this may include the following four categories:

- **Usage** Each agent knows the values it currently has selected in each search context.
- **Support** Each agent can store for each search context, currently known values of other agents (agent-view) and the constraints that need to be satisfied with these values.
- **Nogoods** Each agent can store for each search context, partial assignments that are found to be inconsistent.
- **Effort** Each agent knows for each search context how much effort in terms of the number of backtracks it has already invested.

3.5.4 Interpretation of Knowledge

The interpretation of this knowledge can follow two orthogonal principles: **diversity** and **emulation**. Diversity implements the idea of traversing the search space in different parts simultaneously in order not to miss the part in which a solution can be found. The concept of emulation implements the idea of cooperative problem solving, where agents try to combine (partial) solutions in order to make use of work which others have already done.

	diversity	**emulation**
Usage	*minUsed*: the value which is used the least in other searches	*maxUsed*: the value which is used most in other searches
Support	–	*maxSupport*: the value that is most supported by constraints w.r.t., current agent-views
Nogoods	*differ*: the value which is least included in nogoods	*share:* always use nogoods of all searches
Effort	*minBt*: a value which is not the current value of searches with many backtracks	*maxBt*: the current value of the search with most backtracks

Table 3.1: Methods of aggregation.

With these concepts of providing and interpreting knowledge we can define the portfolio of aggregation methods shown in Table 3.1. In each box we provide a name (to be used in the following) and a short description of which value is preferably selected by an agent for a search.

3.5.5 Practical Implementation of the Knowledge Sharing Policies

The implementation of each knowledge sharing policies is rather simple since they only require regular lookups to other contexts in order to make a decision. More concretely,

- *minUsed*, *maxUsed*. Each value of the initial domain of a local variable is associated to a counter. This counter is updated each time a decision for that variable is made in any search context. Each counter takes values between 0 and the number of contexts. For each variable, pointers to the min (resp. max) used variables are incrementally updated. During a decision, *minUsed* selects the value which is the least used in other contexts, while *maxUsed* selects the most used.

- *maxSupport*. Each value of the initial domain of a local variable is associated to a counter. This counter stores the number of supports each value has in other contexts. In order to illustrate this policy, let us consider an example with an inter-agent constraint $X \leq Y$ where X and Y have initial domains $\{a, b, c\}$. Now let us assume that two different agents own the variables, and that the M- framework uses three contexts where, $Y = a$ in the first one, and $Y = b$ in the second one. If the agent owning X has

to decide for its value in the third context, it will have the following values for the maxSupport counters: maxSupport(a)=2, maxSupport(b)=1, maxSupport(c)=0. It will then select the value a since this value is the most supported w.r.t., its current agent-views. Remark that implementing a *minSupport* policy would be straight forward with the previous counters. We did not try that policy, just because it does not really make sense from a problem solving view point.

- *differ*. Each value of the initial domain of a local variable is associated to a counter. This counter is increased each time a nogood which contains a particular value is recorded by ABT in any search context. During a decision, the value with the lowest counter is selected.

- *share*. With this policy, each nogood learnt by ABT is automatically reused in other search contexts.

- *minBt*, *maxBt*. The number of local backtracks performed by the agent in each of the context is recorded. Each time a value has to be selected for a particular variable, *minBt* forces the selection of the value used for the same variable in the search with the least number of backtracks. Inversely, *maxBt* forces the selection of the value used in the search with the largest number of backtracks.

As we can see, even the most complex policies only requires the association of counters to domains values. These counters aggregate information between search contexts at the agent level. They are updated during any decision of any particular context, and used to perform better decisions in any particular context. Updating these counters can be done naively or incrementally, for instance with the help of some bookkeeping technique.

3.5.6 Complexity

Before presenting the empirical evaluation of M-, we discuss its costs hereafter.

Space

The trade-off in space for the application of M- is linear in the number of applied orderings. This is obvious for our implementation (see Figure 3.3). Thus, it clearly depends on the size of the data structures that need to be duplicated for the contexts. This will include only internal data structures which are related to the state of the search. M- does not duplicate the whole agent. For instance, the data structures for communication are jointly used by all the concurrent search efforts as shown in the Figure 3.3.
It turned out in our experiments that this extra space requirement is very small. We observed that the extra memory needed with a portfolio of size ten applied to IDIBT is typically only about 5–10%. For ABT the extra memory when using 10 instead of one context differed depending on the problem. For easy problems,

where few nogoods need to be stored the extra memory consumption was about 5–20%. For hard problems we could observe up to 1000% more memory usage of the portfolio. This clearly relates to the well-known space-trade-off of nogood-recording.

Network load

The trade-off in network load, that is the absolute number of messages is linear in the portfolio size. When using M parallel contexts that perform one search effort each we will in the worst case have M times more messages. However, on average this may be less because not all of the M searches will terminate. As soon as one has found a solution the complete system will stop and $M-1$ search efforts will omit the rest of their messages.
Furthermore, the absolute number of messages is not the most crucial metric in DisCSP. As described earlier sequential messages are more appropriate. The sequential messages do not increase in complexity because the parallel search efforts are independent of one each other such that the number of sequential messages (smc) is the maximum of the smc of all searches in the worst case. In average, however, it will be the smc of the search that is best. Consequently, the smc-complexity when using M-X is the same as the smc-complexity of X.
Using Aggregation will not increase the number of required messages because this is performed internally by the agents.

Algorithm Monitoring

The complexity of monitoring M-X is the same as it is necessary for the algorithm X. This includes starting the agents and termination detection. Since the number of agents is not increased when using M- we do not need any extra communication or computation for these tasks.

Time

The trade-off in computational costs increases with the use of M-. Similar to the increase in absolute messages we have a linear increase in constraint checks. However, looking at non-concurrent constraint checks (nccc) the complexity of X and M-X is the same provided there is no aggregation. The derivation of this conclusion can be made analogously to the derivation concerning smc.
When we use Aggregation, however, there may be an increase in computational costs of the agents. Depending on the effort an agent puts in using information it gets from other contexts this may also increase the number of nccc. This will be analyzed in the next Section.
Therefore, the overall cost of M-X is the same as the worst-case complexity of X when we use the concurrent metrics. In average, however, M- will be "as good as the best search heuristic" or even "better than the best" when knowledge sharing techniques are implemented. This will be presented in the next section.

3.6 Empirical Evaluation

For the empirical evaluation of the M-framework we processed more than 180,000 DisCSPs with M-IDIBT and M-ABT. We solved random binary problems (15 variables, 5 values), n-queens-problems with n up to 20 and quasigroup completion problems with up to 81 agents.
All tests were run in a Java multi-threaded simulator where each agent implements a thread. The common memory of the whole process was used to implement message channels. Agents can send messages to channels where they are delayed randomly for 1 to 15 milliseconds. This was done to simulate real world contingencies in messages deliveries. After this delay they may be picked up by their addressee. All threads have the same priority such that we have no influence on the activation of them and on the computational resources assigned to them by the JVM or the operating system.
In this simulator we implemented the metrics described in Section 3.3.3. The absolute numbers of messages (mc), constraint checks (cc) and backtracks (bt) were counted locally and accumulated after termination of the algorithm. The more sophisticated metrics which reflect the parallelism were computed during the execution of the algorithms. Whenever a message is passed from A to B then A will include its current value of nccc and smc. The receiver takes the maximum of the value and its locally stored values, adds the costs it is now performing and passes the result on with the next message it sends. After termination of the algorithm we select the maximum of all these values among all agents. Remark that there have been recent research which tried to define alternative performance metrics for DisCSP and DCOP (optimization) problems (see [SLS+08, GZG+08]).

3.6.1 Basic Performance

In Figure 3.4 we show the median[1] numbers of messages sent and the runtime to find one solution by different sized portfolios on fairly hard instances (density 0.3, tightness 0.4) of random problems (sample size 300). These problems represent randomly generated CSPs where the link density between variables is set to 30%, whereas the tightness density of each constraint is set to 40%, i.e., 40% of the values combinations for the underlying constraint are disabled. No aggregation was used in these experiments. The best known[2] variable-ordering (maxDegree) was used in each portfolio including those of size 1 which are equivalent to the basic algorithms. In the larger portfolios we added instances of lex, random and minDegree and further instances of all four added in this order. For example 6-ABT would use the orders (maxDeg, lex, rand, minDeg, maxDeg, lex). It can be seen that with increasing portfolio-size there is more communication between agents. The absolute number of messages rises. In the

[1] We decided to use the median instead of the mean to alleviate the effects of messages interleaving. Indeed, interleaving can give disparate measures which can be pruned by the median calculation.

[2] We made preliminary experiments to determine this.

same Figure we show the runtime. It can be seen that the performance improves up to a certain point when larger portfolios are used. In our experimental setting this point is reached with size 10. With larger portfolios no further speedup can be achieved which would make up the communication cost and computational overhead. The same behaviour can be observed when considering smc or nccc.

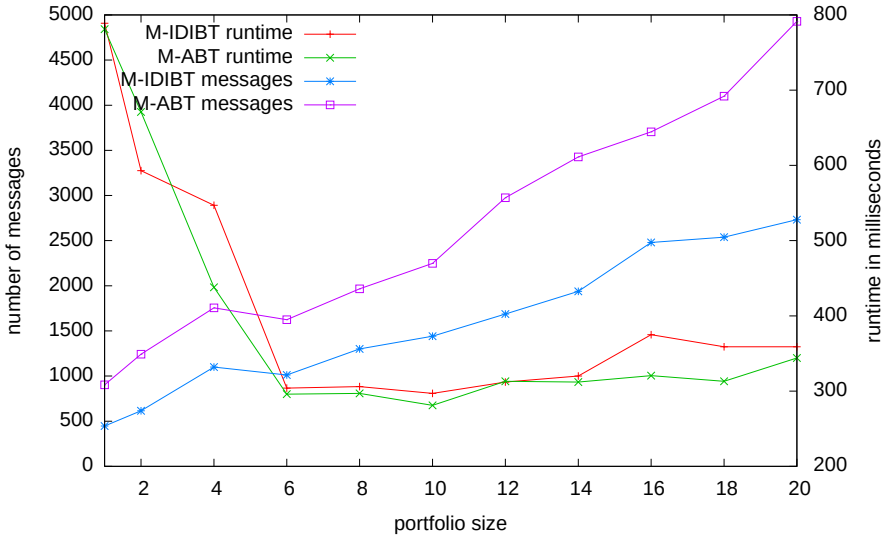


Figure 3.4: Communication and runtime in M- portfolios.

3.6.2 Randomization Risk

The Randomization-Risk is defined as the standard deviation within each sample in our experimental setup. To evaluate it we applied M-IDIBT with homogeneous portfolios 30 times each to a set of 20 hard random problem instances $< 15, 5, 0.3, 0.5 >$. All portfolios used the same deterministic value selection function and variable ordering (both lexicographic) in all searches. For each problem instance we considered the standard deviation of the 30 runs. Then we took the average of these standard deviations over all 20 problem instances for each portfolio size. This would give us the R-Risk that emerges exclusively from the distribution. The results for portfolios size 1 to 8 can be seen in Figure 3.5. It can be seen that all three relevant performance measures (nccc, smc, and pt) decrease with portfolio size increased from 1 to 2. This means the randomization risk decreases when we apply the M-framework. Beyond two there is only a slight decrease.

In order to check the influence of the M-framework to the heavy-tail behaviour we repeated the experiment described in Section 3.4.1 (quasigroup completion of order 6 for ABT and order 7 for IDIBT with 42% preassigned values, sample size

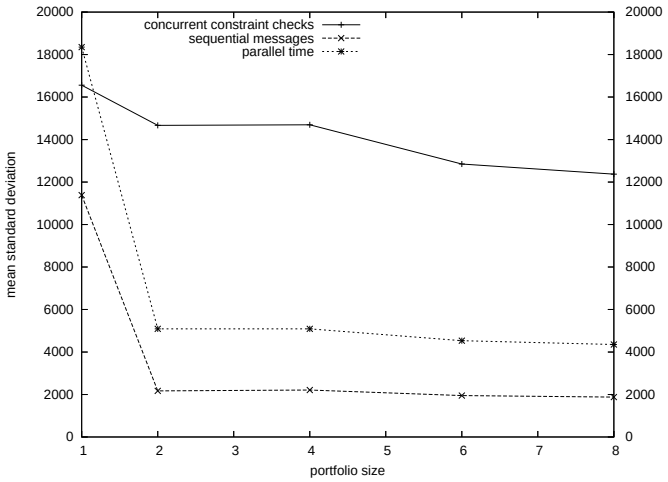


Figure 3.5: Randomization-Risk emerging from message delays and thread activation.

800) with portfolios of size 10. In Figure 3.6 we show the cumulated distribution function of the absolute number of backtracks when applying M-ABT and M-IDIBT to the quasigroup completion problems on a log-log scale. It can be seen that both curves decrease in more than a linear manner. As described earlier this implies the non-heavy-tailedness of the runtime distribution of these algorithms.

3.6.3 Selection-Risk

To evaluate the selection-risk we used a similar experimental setting as before but with heterogeneous variable orderings in the portfolios. We chose to use M different random variable orderings in a portfolio of size M. This would reduce the effects we get from knowledge about variable selection heuristics. The value selection was the same (lexicographic) in all experiments in order to reduce the portion of R-Risk as widely as possible and to expose the risk emerging from the selection of a particular variable ordering. In this setting we would get an unbiased evaluation of the risk we take when choosing variable orderings. The mean standard-deviation of the parallel runtime for M-ABT and M-IDIBT is shown in Figure 3.7 on a logarithmic scale. It can be seen that the risk is reduced significantly with the use of portfolios. With portfolio size 20, for instance, the S-risks of M-IDIBT and M-ABT are 344 and 727 times smaller than the ones of IDIBT and ABT, respectively.

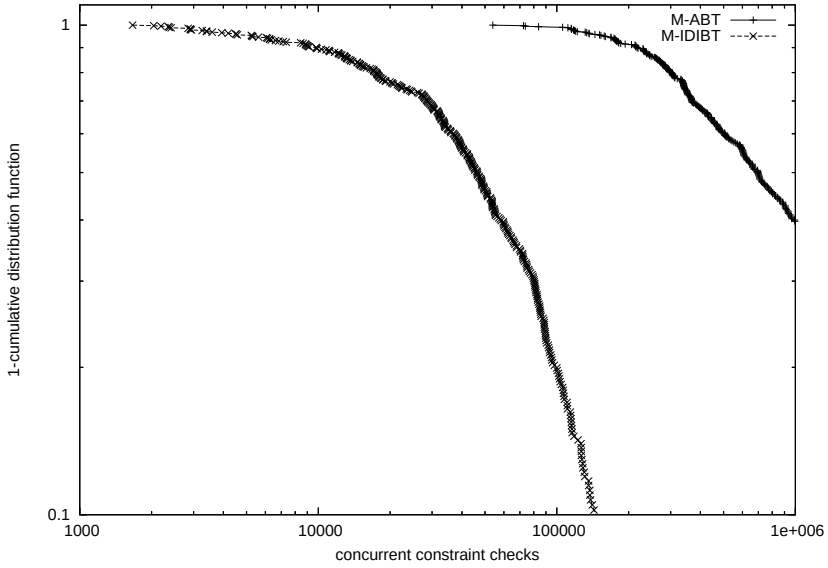


Figure 3.6: No heavy-tails with M-ABT and M-IDIBT.

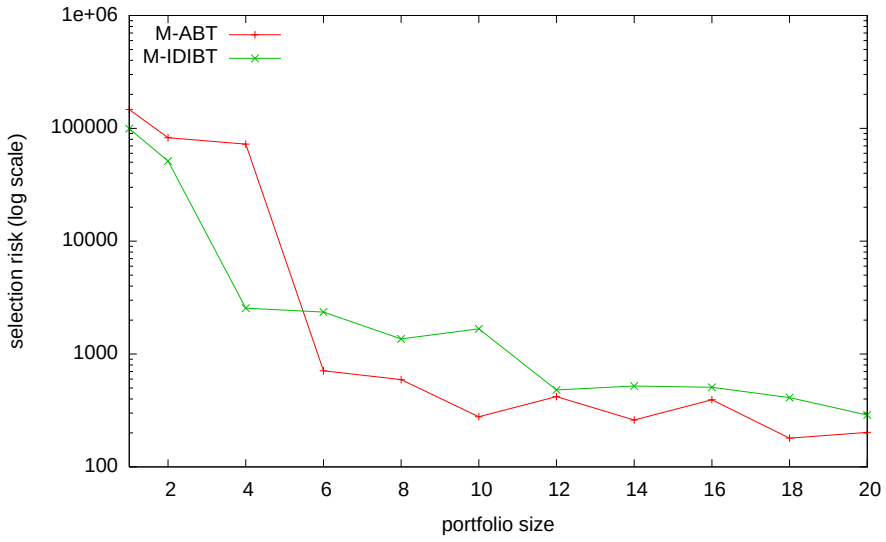


Figure 3.7: S-Risk (standard-dev of the parallel runtime) including the R-Risk emerging from distribution.

3.6.4 Performance with Aggregation

The benefit of Aggregation which is implemented with the different value selection heuristics is presented in Table 3.2. Each column in the table shows the median values of at least 100 samples solved with M-IDIBT with a portfolio of size 10 applied to 30 different hard random and quasigroup completion problems.

	hard randoms			quasigroups		
	smc	nccc	pt	$\frac{smc}{1000}$	$\frac{nccc}{1000}$	pt
minUsed	367	2196	1.563	102	1625	448
maxUsed	379	**2118**	**1.437**	40	**635**	182
minBt	392	2281	1.640	104	1330	367
maxBt	433	2541	1.820	43	694	171
maxSupp	**57**	5718	1.922	**1.9**	3727	**143**
random	409	2406	1.664	73	1068	298

Table 3.2: Performance of aggregation methods for M-IDIBT.

In the table we refer to the aggregation methods introduced in Table 3.1, the bottom line shows the performance with random value selection (and thus no aggregation). When we consider the parallel runtime, it seems that the choice of the best method depends on the problem. For the quasigroup, aggregation

based on the emulation principle seems to be better, on random problems not. Interestingly, message passing operations present a different picture. It can be seen that *maxSupport* uses by far the least messages. These operations are reduced by a factor of 7 (resp. 38) for random (resp. quasigroups) problems. However, when we consider parallel time, it cannot outperform the others significantly since our implementation of this aggregation method is relatively costly[3]. However, message passing is the most critical operation in real systems because of either long latencies or high energy consumption (e.g., ad-hoc networks [FM02]). This makes the *maxSupport* aggregation method really promising. Indeed, there is a clear correlation between the amount of messages sent and the amount of local computations. Especially in DisCSPs where agents are hosting complex sub-problems. In these situations, since every incoming message may trigger the search of a new solution for the local problem, it is important to restrict message passing.
The performance of *maxSupport* can be explained as follows. It benefits from the efforts done in other contexts by capitalizing on compatible values i.e., support relation. As a result this aggregation strategy effectively **mixes the partial solutions constructed in the different contexts**. It corresponds to an effective juxtaposition of partial solutions.

3.6.5 Scalability

In order to evaluate the relevance of the M-framework we investigated how it scales in larger and more structured problems. For this we applied good configurations found in the previous experiments to the quasigroup completion problem as described earlier in Section 3.4.1 (straightforward modelling with binary constraints, most difficult instances with 42% pre-assignment).
Table 3.3 shows the experimental results of distributed search algorithms on problems of different orders (each column represents an order). ABT and IDIBT used the domain/degree (domDeg) variable ordering [BR96], which was tested best in preliminary experiments. In the larger portfolios we used domain/degree and additional heuristics including maxDegree, minDomain, lex and random. In all portfolios Aggregation with the method *maxUsed* was applied[4]. For each order (column) we show the median parallel runtime (in seconds) to solve 20 different problems (once each) and the number of solved problems. When less than 10 instances could be solved within a timeout of two hours we naturally cannot provide meaningful median results. In the experiments with M-ABT we have also observed runs which were aborted because of memory problems in our simulator. For order 8 these were about one third of the unsolved problems, for order 9 this problem occurred in all unsuccessful tests. This memory problem arising from the nogood-storage of ABT was addressed in [BBMM05] and is not the subject of this research.

[3] It seems that bookkeeping could help to reduce the amount of constraint checks.
[4] We decided to use this method since it was shown to minimize nccc on previous tests (see Table 3.2).

	5	6	7	8	9
ABT	**0.3, 20**	-, 8	-, 1	-, 0	-, 0
M-ABT, size 5	0.5, 20	5.9, 19	35.8, 14	-, 2	-, 0
M-ABT, size 10	0.6, 20	**6.1**, 20	**40.6**, 17	-, 8	-, 1
IDIBT	1.8, 20	12.4, 20	234, 20	4356, 16	-, 5
M-IDIBT, size 5	**0.2, 20**	**0.9, 20**	9.3, 20	709, 20	-, 6
M-IDIBT, size 10	0.3, 20	1.7, 20	**8.2, 20**	**339, 20**	-, 8

Table 3.3: Median parallel runtime (pt) and instances solved (out of 20) of quasigroup completion problems with 42% pre-assigned values.

From the successful tests it can be seen that portfolios improve the median performance of IDIBT significantly. In the problems of order 7 a portfolio of 10 was 28 times faster than the regular IDIBT. Furthermore, portfolios seem to become more and more beneficial in larger problems as the portfolio of size 10 seems to scale better than the smaller one. ABT does not benefit in the median runtime but the reduced risk makes a big difference. With the portfolio of size 10, we could solve 17 instances of order 7 problems whereas the plain algorithm could only solve one.

3.6.6 Idle Time

To complete the presentation of our experimental results let us consider time utilization in distributed search. It appears that agents in both considered classical algorithms under use available resources. This is figured in the first two columns of Table 3.4 for various problem classes. The numbers represent the average idle times (10-100 samples) of the agents. In our simulator we captured the idle times in each agent separately. Each agent accumulates the time it waits for new messages to be processed. Whenever an agent finished processing one message and has no new message received it starts waiting until something arrives in its message channel. This waiting time is accumulated locally. After termination of the algorithm we take the mean of these accumulated times of all agents to compute the numbers shown in Table 3.4.

problem class	idle time of agents			
	ABT	IDIBT	M-ABT	M-IDIBT
easy random	87%	92%	56%	47%
hard random	92%	96%	39%	59%
n-queens	91%	94%	48%	52%
hard quasigroups	87%	93%	28%	59%

Table 3.4: Idle times of agents in DisCSP.

We can observe that ABT (Asynchronous BackTracking) and IDIBT (Inter-

leaved Distributed Intelligent BackTracking) are most of the time idle. This idleness comes from the inherent disbalance of work in DisCSPs. Indeed, it is well known that the hierarchical ordering of the agents makes low priority agents (at the bottom) more active than high priority ones. Ideally the work should be balanced. Thus, ideally one agent on the top of the hierarchy in context 1 should be in the bottom in context 2, etc (e.g., see agent in charge of variable X_1 in figure 3.2). Obviously, since we use well known variable ordering heuristics we cannot enforce such a property. However, the previous is an argument for M- which can use idle time "for free" in order to perform further computations in concurrent search efforts. This effect is shown in the last two columns of the table where the M-framework with a portfolio of size 10 applied to the same problems. These algorithms make a better use of computational resources. Certainly it is not a goal to reduce idleness to a minimum since the performance of our algorithm also depends in the response-times of the agents which may become very long with low idleness. However, without having studied this intensively we are convinced that a mean idleness of more than 90% is not necessary for fast responses.

3.7 Summary

We have presented a generic framework for the execution of DisCSP algorithms. It was tested on two standard methods but any tree-based distributed search should easily fit in the M-framework. The framework executes a portfolio of cooperative DisCSP algorithms with different agent-orderings concurrently until the first of them terminates. In real (truly distributed) applications, our framework will have to start with the computation of different orderings. The generic Distributed Agent Ordering heuristic (DisAO) [HBQ98] could easily be generalized at no extra message passing cost to concurrently compute several distributed hierarchies. The main idea is to simultaneously exchange multiple heuristic evaluation of a sub-problem instead of one.

Heterogeneous portfolios are shown to be very beneficial. They improve the performance and reduce the risk in distributed search. With our framework we were able to achieve a speedup of one order of magnitude while reducing the risk by up to three orders of magnitude compared to the traditional execution of the original algorithm. The chances of extensive thrashing due to bad early decisions (so called heavy-tails) are significantly diminished.

A portfolio approach seem to make a better use of computational resources by reducing the idle time of agents. This is the first of two special advantages of the application of portfolios in DisCSP: we do not have to artificially add parallelism and the related overhead but can use idle resources instead. The M-framework can be seen as a solution to the classical "work imbalance" flaw of tree-based distributed search.

We analyzed and defined distributed cooperation (Aggregation) with respect to two orthogonal principles *diversity* and *emulation*. Each principle was applied without overhead within the limited scope of each agent's knowledge. This is

the second special advantage of using portfolios in DisCSP: aggregation yields no communication costs and preserves privacy ([GGS07]) because processes are not related to search efforts but to agents instead. Our experiments identified the emulation-based *maxSupport* heuristic as the most promising one. It is able to efficiently aggregate partial solutions which results in a large reduction in message passing operations.
In the next chapter we will see that the ideas developed here can be applied in the context of parallel satisfiability.

Chapter 4

Parallel Satisfiability

The work developed in this Chapter was initiated in Cambridge, and extended during the co-supervision of the PhD thesis of Said Jabbour and Long Luo (CRIL-CNRS), and the supervision of the post-doc of Said Jabbour (MSR-INRIA). It also led to two MSR summer internships (Said Jabbour, and Christoph Wintersteiger). It produced the following publications and US patent application:

- Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT: solver description. Technical Report MSR-TR-2008-83, Microsoft Research, May 2008.
- Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT: a parallel SAT solver. *JSAT*, 6(4):245–262, 2009.
- Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Control-based clause sharing in parallel SAT solving. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 499–504, 2009.
- Lucas Bordeaux, Youssef Hamadi, and Horst Samulowitz. Experiments with massively parallel constraint solving. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 443–448, 2009.
- Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. A concurrent portfolio approach to SMT solving. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 715–720. Springer, 2009.
- Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Diversification and intensification in parallel SAT solving. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2010.
- Youssef Hamadi, Said Jabbour, Cédric Piette, and Lakhdar Sais. Deterministic parallel DPLL. *JSAT*, 7(4):127–132, 2011.
- Youssef Hamadi, João Marques-Silva, and Christoph M. Wintersteiger. Lazy decomposition for distributed decision procedures. In Jiri Barnat and Keijo Heljanko, editors, *PDMC*, volume 72 of *EPTCS*, pages 43–54, 2011.
- Alejandro Arbelaez and Youssef Hamadi. Improving parallel local search for SAT. In Carlos A. Coello Coello, editor, *LION*, volume 6683 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2011.
- Youssef Hamadi and Christoph M. Wintersteiger. Seven challenges in parallel SAT solving. In *AAAI, Invited paper, to appear*, 2012.

- Youssef Hamadi and Christoph M. Wintersteiger. Challenges in parallel SAT solving. In *AI Magazine, Invited article, to appear*, 2012.
- US Patent App. No. 12465440, Controlled Constraint Sharing in Parallel Problem Solvers, Youssef Hamadi and Said Jabbour, May 13, 2009.

It also led to the following awards:

- SAT-Race 2008 : Gold medal for the ManySAT solver with title of *Best Parallel SAT Solver*. SAT'08, Guangzhou, P. R. China.
- SAT-Competition 2009: ManySAT ranked first, second, and third, industrial category. SAT'09, Swansea, United Kingdom.
- SAT-Race 2010: Silver and Bronze medals for the ManySAT solver. SAT'10, Edinburgh, United Kingdom.
- SAT-Competition 2011: CSLS solver, silver medal SAT+Random category. SAT'11 Ann Arbor, USA.

Finally, it was presented in numerous invited talks and tutorials (see Part II).

4.1 Introduction

In the previous chapter, we have seen how a portfolio of algorithms, opportunistically exchanging knowledge about the problem can be used to boost the performance of distributed search by several orders of magnitude. In this chapter, we are going to apply the same concepts to centralized search, i.e., to situations where the problem is fully expressed in one particular node or agent. More specifically, we are going to apply parallel portfolios to the important domain of propositional satisfiability.

In recent years, SAT solvers had a huge impact in their traditional hardware and software verification domains. Today, they are also gaining popularity in new fields like Automated Planning, General Theorem Proving or Computational Biology [Rin11, dMB08, CBH+07]. This widespread adoption is the result of the efficiency gains made during the last decade [BHZ06]. Indeed, many industrial problems with hundreds of thousands of variables and millions of clauses are now solved within a few minutes. This impressive progress can be related to both low level algorithmic improvements and to the ability of SAT solvers to exploit the hidden structures of a practical problem.

However, many new applications with instances of increasing size and complexity are coming to challenge modern solvers, while at the same time, it becomes clear that the gains traditionally given by low level algorithmic adjustments are almost gone. As a result, a large number of industrial instances from the last competitions remain challenging for all the available SAT solvers. Fortunately, the previous comes at a time where the generalization of multicore hardware gives parallel processing capabilities to standard PCs. While in general it is

important for existing applications to exploit new hardwares, for SAT solvers, this becomes crucial.
Many parallel SAT solvers have been previously proposed. Most of them are based on the divide-and-conquer principle (see Section 4.2). They either divide the search space using for example guiding paths or the formula itself using decomposition techniques. The main problem behind these approaches rises in the difficulty to get workload balanced between the different processor units or workstations. Another drawback of these approaches rises in the fact that for a given large SAT instance with hundreds of thousands of variables it is very difficult to find the most relevant set of variables to divide the search space.
In the following, we detail ManySAT, a new parallel SAT solver, winner of the 2008 Sat-Race[1]. The design of ManySAT takes advantage of the main weakness of modern solvers: their sensitivity to parameter tuning. For instance, changing the parameters related to the restart strategy or to the variable selection heuristic can completely change the performance of a solver on a particular problem class. In a multicore context, we can easily take advantage of this lack of robustness by designing a portfolio which will run different incarnations of a sequential solvers on the same instance. Each solver would exploit a particular parameter set and their combination should represent a set of orthogonal yet complementary strategies. Moreover, individual solvers could perform knowledge exchange in order to improve the performance of the system beyond the performance of its individual components.
As we can see, the ManySAT approach is a direct application of our previous M- framework to SAT. Unlike in M-, ManySAT solves centralized problems and use multiple resources to speed-up processing. Here M- search-context correspond to proper execution of a sequential SAT engine. In the portfolio, engines are not only differentiated with respect to their labelling strategies but to various other features of SAT solvers.

4.2 Previous Work

We present here the most noticeable approaches related to parallel SAT solving.
PSATO [ZBH96] is based on the SATO (SAtisfiability Testing Optimized) sequential solver [ZS94]. Like SATO, it uses a *trie* data structure to represent clauses. PSATO uses the notion of *guiding-paths* to divide the search space of a problem. These paths are represented by a set of unit clauses added to the original formula. The parallel exploration is organized in a master/slave model. The master organizes the work by addressing guiding-paths to workers which have no interaction with each others. The first worker to finish stops the system. The balancing of the work is organized by the master.
In [JLU05] a parallelization scheme for a class of SAT solvers based on the DPLL procedure is presented. The scheme uses a dynamic load-balancing mechanism based on work-stealing techniques to deal with the irregularity of SAT problems. PSatz is the parallel version of the well known Satz solver.

[1] http://www-sr.informatik.uni-tuebingen.de/sat-race-2008/index.html

Gradsat [CW03] is based on zChaff. It uses a master-slave model and the notion of guiding-paths to split the search space and to dynamically spread the load between clients. Learned clauses are exchanged between all clients if they are smaller than a predefined limit on the number of literals. A client incorporates a foreign clause when it backtracks to level 1 (top-level).
In [BSK03], the authors use an architecture similar to Gradsat. However, a client incorporates a foreign clause if it is not subsumed by the current guiding-path constraints. Practically, clause sharing is implemented by *mobile-agents*. This approach is supposed to scale well on computational grids.
Nagsat [FS02] is a parallel SAT solver which exploits the heavy-tailed distribution of random 3-SAT instances. It implements *nagging*, a notion taken from the DALI theorem prover. Nagging involves a master and a set of clients called *naggers*. In Nagsat, the master runs a standard DPLL algorithm with a static variable ordering. When a nagger becomes idle, it requests a *nagpoint* which corresponds to the current state of the master. Upon receiving a nagpoint, it applies a transformation (e.g., a change in the ordering or the remaining variables), and begins its own search on the corresponding subproblem.
In [BS96], the input formula is dynamically divided into disjoint subformulas. Each subformula is solved by a sequential SAT-solver running on a particular processor. The algorithm uses optimized data structures to modify Boolean formulas. Additionally workload balancing algorithms are used to achieve a uniform distribution of workload among the processors.
MiraXT [LSB07], is designed for shared memory multiprocessors systems. It uses a divide-and-conquer approach where threads share a unique clause database which represents the original and the learnt clauses. When a new clause is learnt by a thread, it uses a lock to safely update the common database. Read access can be done in parallel.
PMSat uses a master-slave scenario to implement a classical divide-and-conquer search [GFS08]. The user of the solver can select among several partinioning heuristics. Learnt clauses are shared between workers, and can also be used to stop efforts related to search spaces that have been proven irrelevant. PMSat runs on networks of computer through an MPI implementation.
In [CS08], the authors use a standard divide-and-conquer approach based on guiding-paths. However, it exploits the knowledge on these paths to improve clause sharing. Indeed, clauses can be large with respect to some static limit, but when considered with the knowledge of the guiding path of a particular thread, a clause can become small and therefore highly relevant. This allows pMiniSat to extend the sharing of clauses since a large clause can become small in another search context.

4.3 Technical Background

In this section, we first recall the basis of the most commonly used DPLL search procedure. Then, we introduce some computational features of modern SAT solvers. A brief description of multicore based architectures is given. Finally,

we present the principle of the AIMD feedback control-based algorithm used by advanced versions of ManySAT to manage knowledge sharing.

4.3.1 DPLL Search

Most of the state of the art SAT solvers are simply based on the Davis, Putnam, Logemann and Loveland procedure, commonly called DPLL [DLL62]. DPLL is a backtrack search procedure; at each node of the search tree, a *decision* literal is chosen according to some branching heuristics. Its assignment to one of the two possible values (true or false) is followed by an *inference* step that deduces and propagates some forced literal assignments such as unit and monotone literals. The assigned literals (decision literal and the propagated ones) are labeled with the same decision level starting from 1 and increased at each decision (or branching) until finding a model or reaching a conflict. In the first case, the formula is answered to be satisfiable, whereas in the second case, we backtrack to the last decision level and assign the opposite value to the last decision literal. After backtracking, some variables are unassigned, and the current decision level is decreased accordingly. The formula is answered to be unsatisfiable when a backtrack to level 0 occurs. Many improvements have been proposed over the years to enhance this basic procedure, leading now to what is commonly called modern SAT solvers. We also mention that, some look-ahead based improvements are at the basis of other kind of DPLL SAT solvers (e.g. Satz [LA97], kcnfs [DD01], march-dl [HvM06]) particularly efficient on hard random and crafted SAT categories.

4.3.2 Modern SAT Solvers

Modern SAT solvers [MMZ$^+$01, ES03a], are based on classical DPLL search procedure [DLL62] combined with (i) restart policies [GSK98, KHR$^+$02], (ii) activity-based variable selection heuristics (VSIDS-like) [MMZ$^+$01], and (iii) clause learning [MSS96]. The interaction of these three components being performed through efficient data structures (e.g., watched literals [MMZ$^+$01]). All the state-of-the-art SAT solvers are based on a variation in these three important components.
Modern SAT solvers are especially efficient with "structured" SAT instances coming from industrial applications. VSIDS and other variants of activity-based heuristics [BGS99], on the other hand, were introduced to avoid thrashing and to focus the search: when dealing with instances of large size, these heuristics direct the search to the most constrained parts of the formula. Restarts and VSIDS play complementary roles since the first component reorder assumptions and compacts the assumptions stack while the second allows for more intensification. Conflict Driven Clause Learning (CDCL) is the third component, leading to non-chronological backtracking. In CDCL a central data-structure is the *implication graph*, which records the partial assignment that is under construction together with its implications [MSS96]. Each time a dead end is encountered (say at level i) a conflict clause or nogood is learnt due to a bottom up traversal of the

implication graph. This traversal is also used to update the activity of related variables, allowing VSIDS to always select the most active variable as the new decision point. The learnt conflict clause, called asserting clause, is added to the learnt data base and the algorithm backtracks non chronologically to level $j < i$.
Progress saving is another interesting improvement, initially introduced in [FD94] it was recently presented in the Rsat solver [PD07]. It can be seen as a new selection strategy of the literal polarity. More precisely, each time a backtrack occurs from level i to level j, the literal polarity of the literals assigned between the two levels are saved. Then, such polarity is used in subsequent search tree. This can be seen as a partial component caching technique that avoids solving some components multiple times.
Modern SAT solvers can now handle propositional satisfiability problems with hundreds of thousands of variables or more. However, it is now recognised (see the recent SAT competitions) that the performances of the modern SAT solvers evolve in a marginal way. More precisely, on the industrial benchmarks category usually proposed to the annual SAT-Race and/or SAT-Competitions, many instances remain open (not solved by any solver within a reasonable amount of time). These problems which cannot be solved even using a 3 hours time limit are clearly challenging to all the available SAT solvers. Consequently, new approaches are clearly needed to solve these challenging industrial problems.

4.3.3 Multicore Architectures

We can abstract a multicore architecture as a set of processing units which communicate through a shared memory. In theory, access to the memory is uniform, i.e., can be done simultaneously. Practically, the use of cache mechanisms in processing units creates coherence problems which can slow down the memory accesses.
Our work is built on this shared-memory model. The communication between the DPLLs solvers of a portfolio is organized through lockless queues that contain the lemmas that a particular core wants to exchange.

4.3.4 AIMD Feedback Control-based Algorithm

The Additive Increase/Multiplicative Decrease (AIMD) algorithm is a feedback control algorithm used in TCP congestion avoidance. The problem solved by AIMD is to guess the communication bandwidth available between two communicating nodes. The algorithm performs successive probes, increasing the communication rate w linearly as long as no packet loss is observed, and decreasing it exponentially when a loss is encountered. More precisely, the evolution of w is defined by the following $AIMD(a, b)$ formula:

- $w = w - a \times w$, if loss is detected
- $w = w + \frac{b}{w}$, otherwise

Different proposals have been made in order to prevent congestion in communication networks based on different numbers for a and b. Today, AIMD is the major component of TCP's congestion avoidance and control [Jac88]. On probe of network bandwidth increasing too quickly will overshoot limits (underlying capacities). On notice of congestion, decreasing too slowly will not be reactive enough.
In the context of ManySAT, it is important to exchange knowledge between solvers. We will see that AIMD-based control policies can be used to achieve a particular throughput or a particular throughput of maximum quality. Since any increase in the size limit can potentially generate a very large number of new clauses, AIMD's slow increase can help us to avoid a quick overshoot of the throughput. Similarly, in case of overshooting, aggressive decrease can help us to quickly reduce clause sharing by a very large amount.

4.4 ManySAT: a Parallel SAT Solver

ManySAT is a parallel portfolio of several DPLL-engines which includes all the classical features like two-watched-literal, unit propagation, activity-based decision heuristics, lemma deletion strategies, and clause learning. In addition to the classical first-UIP scheme [ZMMM01], it incorporates a new technique which extends the implication graph used during conflict-analysis to exploit the satisfied clauses of a formula [ABH+08]. In the following, we describe and motivate for a set of important parameters used to differentiate the different solvers in the portfolio.

4.4.1 Restart Policies

Restart policies represent an important component of modern SAT solvers. Contrary to the common belief, in SAT restarts are not used to eliminate the heavy tailed phenomena [GSK98, GSCK00] since after restarting SAT solvers dive in the part of the search space that they just left. In SAT, restarts policies are used to compact the assignment stack and improve the order of assumptions.
Different restart policies have been previously presented. Most of them are static, and the cutoff value follows different evolution scheme (e.g. arithmetic, geometric, Luby). To ensure the completeness of the SAT solver, in all these restarts policies, the cutoff value in terms of the number of conflicts increases over the time. The performance of these different policies clearly depends on the considered SAT instances. More generally, rapid restarts (e.g. Luby) perform well on industrial instances, however on hard SAT instances slow restarts are more suitable. Generally, it is hard to say in advance which policy should be used on which problem class [Hua07].
Our objective was to use complementary restart policies to define the restart cutoff x_i.
We decided to use the well known Luby policy [LSZ93], and a classical geometric policy, $x_i = 1.5 \times x_{i-1}$ with $x_1 = 100$ [ES03a]. The Luby policy was used with

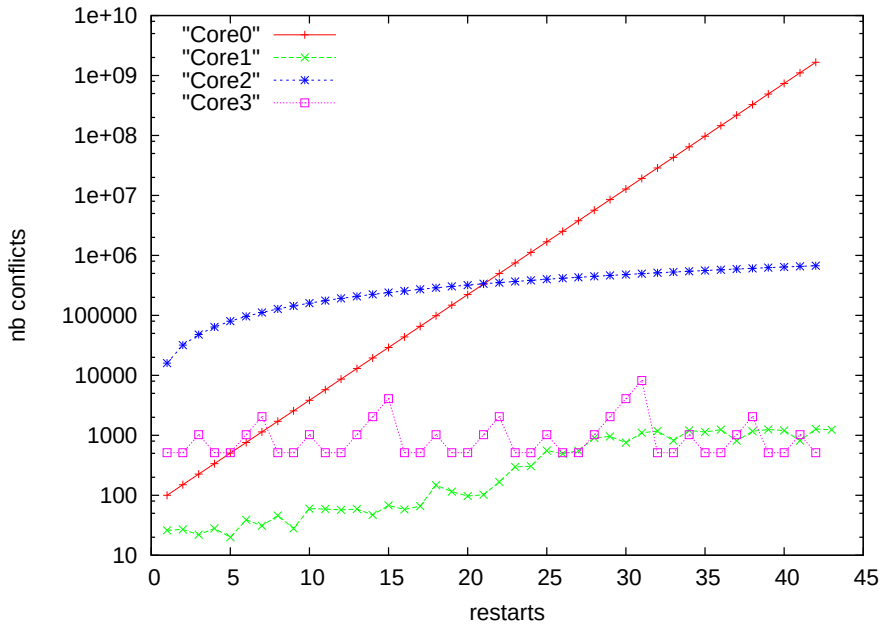


Figure 4.1: Restart strategies

a unit factor set to 512. In addition, we decided to introduce two new policies. A very slow arithmetic one, $x_i = x_{i-1} + 16000$ with $x_1 = 16000$, and a new dynamic one.

New Dynamic Restart Policy

The early work on dynamic restart policy goes back to 2008. Based on the observation that frequent restarts significantly improve the performance of SAT solvers on industrial instance, Armin Biere presents in [Bie08] a novel adaptive restart policy that measures the "agility" of the search process dynamically, which in turn is used to control the restart frequency. The agility measures the average number of recently flipped assignments. Low agility enforces frequent restarts, while high agility tends to prohibit restarts.
In [RS08], the authors propose to apply restarts according to measures local to each branch. More precisely, for each decision level d a counter $c(d)$ of the number of conflicts encountered under the decision level d is maintained. When backtracking to the decision level d occurs, if the value $c(d)$ is greater than a given threshold, the algorithm restarts.
Considering CDCL-based SAT solvers, it is now widely admitted that restarts are an important component when dealing with industrial SAT instances, whereas on crafted and random instances they play a marginal role. More precisely, on industrial (respectively crafted) category, rapid (respectively long) restarts are

more appropriate. It is important to note that on hard SAT instances, learning is useless. Indeed, on such instances, conflict analysis generally leads to a learnt clause which includes at least one literal from the level just before the current conflict level. In other words the search algorithm usually backjumps to the level preceding that of the current conflict. For example, if we consider the well known Pigeon-hole problem, learning from conflicts will produce a clause which includes at least one literal from each level. It is also obvious on this example, that learning does not achieve important backjumps in the search tree. The algorithm usually carries out a chronological backtracking.
In the following, we define a new dynamic restart policy based on the evolution of the average size of backjumps. First, such information is a good indicator of the decision errors made during search. Secondly, it can be seen as an interesting measure of the relative hardness of the instance. Our new policy is designed in such a way that, for high (respectively low) fluctuation of the average size of backjumps (between the current and the previous restart), it delivers a low (respectively high) cutoff value. In other words, the cutoff value of the next restart depends on the average size of backjumps observed during the two previous and consecutive runs. We define it as, $x_1 = 100, x_2 = 100$, and $x_{i+1} = \frac{\alpha}{y_i} \times |cos(1 - r_i)|$, $i \geq 2$ where $\alpha = 1200$, y_i represents the average size of backjumps at restart i, $r_i = \frac{y_{i-1}}{y_i}$ if $y_{i-1} < y_i$, $r_i = \frac{y_i}{y_{i-1}}$ otherwise. The cutoff value x_i is minimal when the ratio between the average size of jumps between the two previous and consecutive runs is equal to one.
From the figure 4.1, we can observe that the cutoff value in terms of the number of conflicts is low in the first restarts and high at the last ones. This mean that the fluctuation between two consecutive restarts is more important at the beginning of the resolution process. Indeed, the activity of the variables is not sufficiently accurate in the first restarts, and the sub-problem on which the search focuses is not sufficiently circumscribed.
The dynamic restart policy, presented in this section is implemented in the first version of ManySAT [HJS08] presented at the parallel track of the SAT Race 2008.

4.4.2 Heuristic

We decided to increase the random noise associated to the VSIDS heuristic [MMZ+01] of core 0 since its restart policy is the slowest one. Indeed, that core tends to intensify the search, and slightly increasing the random noise allows us to introduce more diversification.

4.4.3 Polarity

Each time a variable is chosen, one needs to decide if such a variable might be assigned true (positive polarity) or false (negative polarity). Different kinds of polarity have been defined. For example, Minisat usually chooses the negative polarity, whereas Rsat uses progress saving. More precisely, each time a backtrack occurs, the polarity of the assigned variables between the conflict and the

backjumping level are saved. If one of these variables is chosen again its saved polarity is preferred. In CDCL based solvers, the chosen polarity might have a direct impact on the learnt clauses and on the performance of the solver.
The polarity of the core 0 is defined according to the number of occurrences of each literal in the learnt data base. Each time a learnt clause is generated, the number of occurrences of each literal is increased by one. Then to maintain a more constrained learnt data base, the polarity of l is set to $true$ when $\#occ(l)$ is greater than $\#occ(\neg l)$; and to $false$ otherwise. For example by setting the polarity of l to $true$, we bias the occurrence of its negation $\neg l$ in the next learnt clauses.
This approach tends to balance the polarity of each literal in the learnt data base. By doing so, we increase the number of possible resolvents between the learnt clauses. If the relevance of a given resolvent is defined as the number of steps needed to derive it, then a resolvent between two learnt clauses might lead to more relevant clauses in the data base.

As the restart strategy in core 0 tends to intensify the search, it is important to maintain a learnt data base of better quality. However, for rapid restarts as in the core 1 and 3, progress saving is most suitable in order to save the work accomplished. For the core 2, we decided to apply a complementary polarity ($false$ by default as in Minisat).

4.4.4 Learning

Learning is another important component which is crucial for the efficiency of modern SAT solvers. Most of the known solvers use similar CDCL approaches associated with the first UIP (Unique Implication Point) scheme.
In our parallel SAT solver ManySAT, we used a new learning scheme obtained using an extension of the classical implication graph [ABH+08]. This new notion considers additional arcs, called inverse arcs. These are obtained by taking into account the satisfied clauses of the formula, which are usually ignored by classical conflict analysis. The new arcs present in our extended graph allow us to detect that even some decision literals admit a reason, something which is ignored when using classical implication graphs. As a result, the size of the backjumps is often increased.
Let us illustrate this new extended conflict analysis using a simple example. We assume that the reader is familiar with classical CDCL scheme used in modern SAT solvers (see [MSS96, MMZ+01, ABH+08]).
Let $\mathcal{F}$ be a CNF formula and ρ a partial assignment given below : $\mathcal{F} \supseteq \{c_1, \ldots, c_9\}$

(c_1)	$x_6 \vee \neg x_{11} \vee \neg x_{12}$	(c_2)	$\neg x_{11} \vee x_{13} \vee x_{16}$	(c_3)	$x_{12} \vee \neg x_{16} \vee \neg x_2$
(c_4)	$\neg x_4 \vee x_2 \vee \neg x_{10}$	(c_5)	$\neg x_8 \vee x_{10} \vee x_1$	(c_6)	$x_{10} \vee x_3$
(c_7)	$x_{10} \vee \neg x_5$	(c_8)	$x_{17} \vee \neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18}$	(c_9)	$\neg x_3 \vee \neg x_{19} \vee \neg x_{18}$

$\rho = \{\langle \ldots \neg x_6^1 \ldots \neg x_{17}^1 \rangle \langle (x_8^2) \ldots \neg x_{13}^2 \ldots \rangle \langle (x_4^3) \ldots x_{19}^3 \ldots \rangle \ldots \langle (x_{11}^5) \ldots \rangle\}$. The sub-sequence $\langle (x_8^2) \ldots \neg x_{13}^2 \ldots \rangle$ of ρ expresses the set of literals assigned at level

2 with the decision literal mentioned in parenthesis and the set of propagated literals (e.g. $\neg x_{13}$). The current decision level is 5. The classical implication graph $\mathcal{G}^{\rho}_{\mathcal{F}}$ associated to $\mathcal{F}$ and ρ is shown in Figure 4.2 with only the plain arcs. In the sequel, $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the literal $\neg x$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$. Also a clause c subsume a clause c' iff $c \subseteq c'$.
The traversal of the graph $\mathcal{G}^{\rho}_{\mathcal{F}}$ allows us to generate three asserting clauses corresponding to the three possible UIPs (see figure 4.2). Let us illustrate the such resolution process leading to the first asserting clause Δ_1 corresponding to the first UIP.

- $\sigma_1 = \eta[x_{18}, c_8, c_9] = (x^1_{17} \vee \neg x^5_1 \vee \neg x^5_3 \vee x^5_5 \vee \neg x^3_{19})$
- $\sigma_2 = \eta[x_1, \sigma_1, c_5] = (x^1_{17} \vee \neg x^5_3 \vee x^5_5 \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$
- $\sigma_3 = \eta[x_5, \sigma_2, c_7] = (x^1_{17} \vee \neg x^5_3 \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$
- $\sigma_4 = \eta[x_3, \sigma_3, c_6] = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$

As we can see, σ_4 gives us a first asserting clause (that we'll also name Δ_1) because all of its literals are assigned before the current level except one (x_{10}) which is assigned a the current level 5. The intermediate clauses σ_1, σ_2 and σ_3 contain more than one literal of the current decision level 5, and $\neg x_{10}$ is a first UIP. If we continue such a resolution process, we obtain the two additional asserting clauses $\Delta_2 = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee \neg x^3_4 \vee x^5_2)$, corresponding to a second UIP $\neg x^5_2$; and $\Delta_3 = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee \neg x^3_4 \vee x^2_{13} \vee x^1_6 \vee \neg x^5_{11})$, corresponding respectively to a 3rd UIP ($\neg x^5_{11}$) which is the last UIP since it corresponds to the last decision literal in the partial assignment.
In modern SAT solvers, clauses containing a literal x that is implied at the current level are essentially ignored by the propagation. More precisely, because the solver does not maintain the information whether a given clause is satisfied or not, a clause containing x may occasionally be considered by the propagation, but only when another literal y of the clause becomes false. When this happens the solver typically skips the clause. However, in cases where x is true *and all the other literals are false*, an "arc" was revealed for free that could as well be used to extend the graph. Such arcs are those we exploit in our proposed extension.
To explain further the idea behind our extension, let us consider, again, the formula $\mathcal{F}$ and the partial assignments given in the previous example. We define a new formula $\mathcal{F}'$ as follow : $\mathcal{F}' \supseteq \{c_1, \ldots, c_9\} \cup \{c_{10}, c_{11}, c_{12}\}$ where $c_{10} = (\neg x_{19} \vee x_8), c_{11} = (x_{19} \vee x_{10})$ and $c_{12} = (\neg x_{17} \vee x_{10})$

The three added clauses are satisfied under the instantiation ρ. c_{10} is satisfied by x_8 assigned at level 2, c_{11} is satisfied by x_{19} at level 3, and c_{12} is satisfied by $\neg x_{17}$ at level 1. This is shown in the extended implication graph (see Figure 4.2) by the doted edges. Let us now illustrate the usefulness of our proposed extension. Let us consider again the the asserting clause Δ_1 corresponding to

the classical first UIP. We can generate the following strong asserting clause: $c_{13} = \eta[x_8, \Delta_1, c_{10}] = (x_{17}^1 \vee \neg x_{19}^3 \vee x_{10}^5)$, $c_{14} = \eta[x_{19}, c_{13}, c_{11}] = (x_{17}^1 \vee x_{10}^5)$ and $\Delta_1^s = \eta[x_{17}, c_{14}, c_{12}] = x_{10}^5$. In this case we backtrack to the level 0 and we assign x_{10} to *true*.
As we can see Δ_1^s subsumes Δ_1. If we continue the process we also obtain other strong asserting clauses $\Delta_2^s = (\neg x_4^3 \vee x_2^5)$ and $\Delta_3^s = (\neg x_4^3 \vee x_{13}^2 \vee x_6^1 \vee \neg x_{11}^5)$ which subsume respectively Δ_2 and Δ_3.
This first illustration gives us a new way to minimize the size of the asserting clauses.

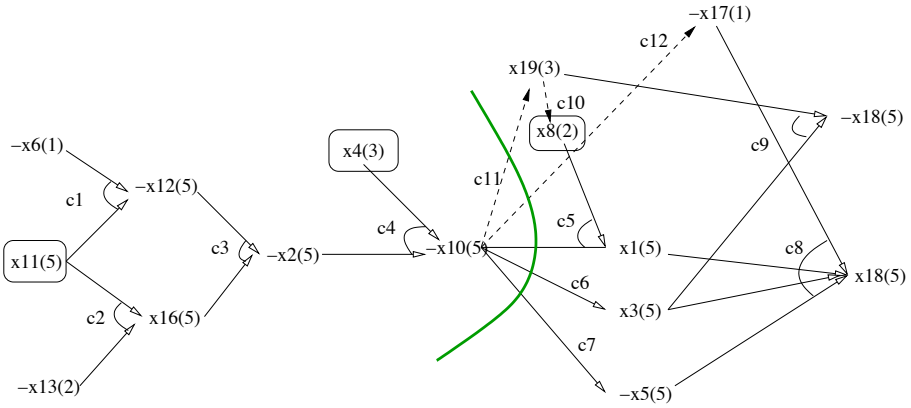


Figure 4.2: Implication graph / extended implication graph

Let us now explain briefly how the extra arcs can be computed. Usually unit propagation does not keep track of implications from the satisfiable sub-formula. In this extension the new implications (deductions) are considered. For instance in the previous example, when we deduce x_{19} at level 3, we "rediscover" the deduction x_8 (which was a choice (decision literal) at level 2). Our proposal keeps track of these re-discoveries.
Our approach makes an original use of inverses arcs to back-jump farther, i.e. to improve the back-jumping level of the classical asserting clauses. It works in three steps. In the first step (1) : an asserting clause, say $\sigma_1 = (\neg x^1 \vee \neg y^3 \vee \neg z^7 \vee \neg a^9)$ is learnt using the usual learning scheme where 9 is the current decision level. As $\rho(\sigma_1) = false$, usually we backtrack to level 7. In the second step (2): our approach aims to eliminate the literal $\neg z^7$ from σ_1 using the new arcs of the extended graph. Let us explain this second and new processing. Let $c = (z^7 \vee \neg u^2 \vee \neg v^9)$ such that $\rho(z) = true$, $\rho(u) = true$ and $\rho(v) = true$. The clause c is an inverse arc i.e. the literal z assigned at level 7 is implied by the two literals u and v respectively assigned at level 2 and 9. From c and σ_1, a new clause $\sigma_2 = \eta[z, c, \sigma_1] = (\neg x^1 \vee \neg u^2 \vee \neg y^3 \vee \neg v^9 \vee \neg a^9)$ is generated. We can remark that the new clause σ_2 contains two literals from the current decision level 9. In the third step (3), using classical learning, one can search from σ_2 for another asserting clause σ_3 with only one literal from the current decision

level. Let us note that the new asserting clause σ_3 might be worse in terms of back-jumping level. To avoid this main drawback, the inverse arc c is chosen if the two following conditions are satisfied : i) the literals of c assigned at the current level (v^9) has been already visited during the first step and ii) all the other literals of c are assigned before the level 7 i.e. level of z. In this case, we guaranty that the new asserting clause achieve better back-jumping.
This new learning scheme is integrated on the SAT solvers of the cores 0 and 3.

4.4.5 Clause Sharing

Unlike in the previously presented M- framework, knowledge in SAT is made of conflict clauses, and knowledge sharing is referred to clause sharing.

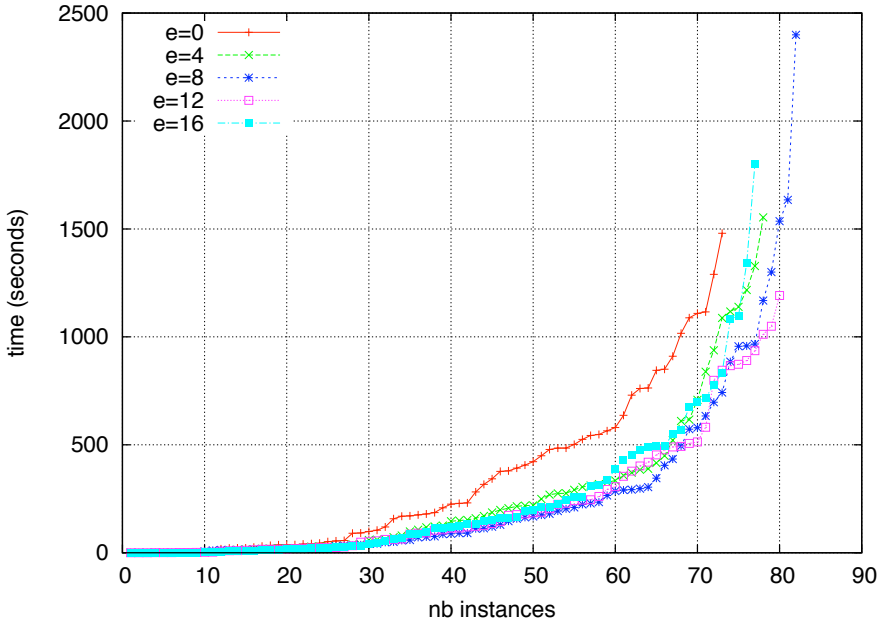


Figure 4.3: SAT-Race 2008: different limits for clause sharing

To start with, we can use a static clause sharing policy where each core exchanges a learnt clause if its size is less or equal to 8. This decision is based on extensive tests with representative industrial instances. Figure 4.3 (respectively Figure 4.4) shows for different limits e the performance of ManySAT on instances taken from the SAT-Race 2008 (respectively SAT-Competition 2007). We can observe that on each set of benchmarks a limit size of 8 gives the best overall performance.
The communication between the solvers of the portfolio is organized through lock-less queues which contain the lemmas that a particular core wants to exchange.

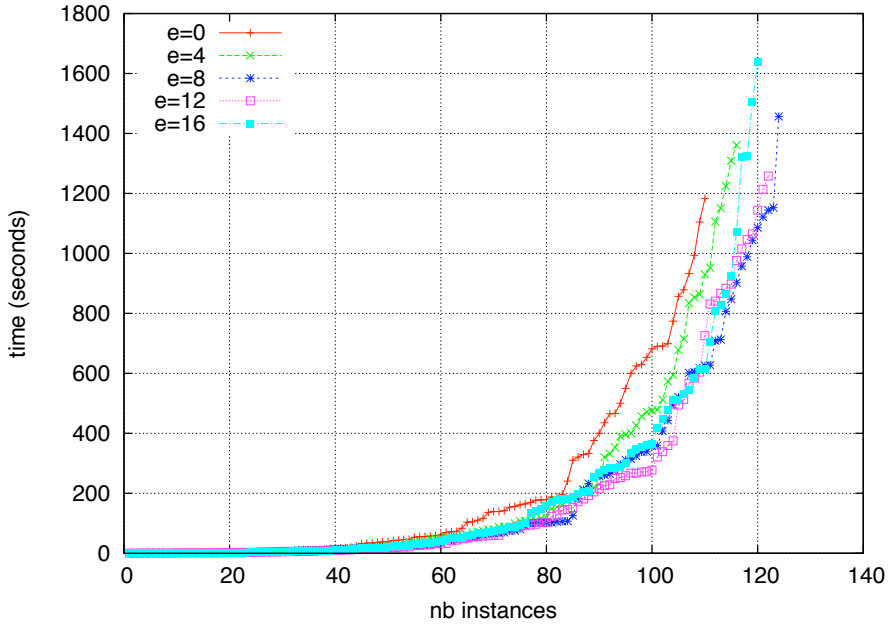


Figure 4.4: SAT-Competition 2007: different limits for clause sharing

Each core imports unit-clauses when it reaches level 0 (e.g., after a restart). These important clauses correspond to the removal of Boolean variables, and therefore are more easily enforced at the top level of the tree.
All the other clauses are imported on the fly, i.e., after each decision. Several cases have to be handled for the integration of a foreign clause c:

- c is false in the current context. In this case, conflict-analysis has to start, allowing the search process to backjump. This is clearly the most interesting case.
- c is unit in the current context. The clause can be used to enforce more unit propagation, allowing the process to reach a smaller fix-point or a conflict.
- c is satisfied by the current context. It has to be watched. To exploit such a clause in the near future, we consider two literals assigned at the highest levels.
- otherwise, c has to be watched. In this last case, the first two unassigned literals are watched.

The following example illustrates the different cases mentioned above.
Let $\mathcal{F}$ be a CNF formula and $\rho = \{\langle \dots \neg x_6^1 \dots \neg x_{17}^1 \rangle \langle (x_8^2) \dots \neg x_{13}^2 \dots \rangle \langle (x_4^3) \dots x_{19}^3 \dots \rangle \dots \langle (x_{11}^5) \neg x_{12}^5, x_{16}^5, \neg x_2^5, \dots, \neg x_{10}^5, x_1^5, \dots, x_{18}^5 \rangle\}$ a partial assignment. To make the

shared clause c exploitable in a near future, it might be watched in a certain way. Suppose that,

- $c = (x_{17}^1 \vee \neg x_{19}^3 \vee x_{10}^5) \in \mathcal{F}$. The clause c is false and the two literals $\neg x_{19}^3$ and x_{10}^5 are watched.
- $c = (x_{17}^1 \vee \neg x_{19}^3 \vee x_{30}) \in \mathcal{F}$. The clause c is unit and the two literals $\neg x_{19}^3$ and x_{30} are watched;
- $c = (x_{17}^1 \vee \neg x_{19}^3 \vee \neg x_{10}^5) \in \mathcal{F}$. We watch the last satisfied literal $\neg x_{10}$ and another literal with the highest level from the remaining ones.
- $c = (x_{25} \vee \neg x_{34} \neg x_{29}) \in \mathcal{F}$. We watch any two literals from c.

4.4.6 Summary

Table 4.1 summarizes the choices made for the different solvers of the ManySAT portfolio. For each solver (core), we mention the restart policy, the heuristic, the polarity, the learning scheme and the size of shared clauses.

4.5 Evaluation

4.5.1 Performance Against a Sequential Algorithm

ManySAT was built on top of Minisat 2.02 [ES03a]. SatElite was applied systematically by each core as a pre-processor [EB05]. In all the figures, instances solved by Satellite in the preprocessing step are not included. In this section, we evaluate the performance of the solver on a large set of industrial problems. Figure 4.5, shows the improvement of performances provided by our solver when opposed to the sequential solver Minisat 2.02 on the problems of the Sat-Race 2008. It shows the performance of ManySAT running with respectively 1,2,3 and 4 cores. When more than one core is used, clause sharing is done up to clause size 8.
We can see that even the sequential version of ManySAT (single core) outperforms Minisat 2.02. This simply means that our design choices for core 1 represent a good combination to put in a sequential solver. Interestingly, with each new core, the performance increases both in speed and number of problems solved. This is the result of the diversification of the search but also the fact that clause sharing quickly boosts these independent search processes.

4.5.2 Performance Against Other Parallel SAT Solvers

We report here the official results of the 2008 Sat-Race. They can be downloaded from the competition website[2]. They demonstrate the performance of ManySAT as opposed to other parallel SAT solvers. These tests were done on 2x Dual-Core Intel Xeon 5150 running at 2.66 GHz, with a timeout set to 900 seconds.

[2] http://www-sr.informatik.uni-tuebingen.de/sat-race-2008/

Strategies	**Core 0**	**Core 1**	**Core 2**	**Core 3**
Restart	Geometric $x_1 = 100$ $x_i = 1.5 \times x_{i-1}$	Dynamic (Fast) $x_1 = 100, x_2 = 100$ $x_i = f(y_{i-1}, y_i), i > 2$ if $y_{i-1} < y_i$ $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \lvert cos(1 - \frac{y_{i-1}}{y_i}) \rvert$ else $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \lvert cos(1 - \frac{y_i}{y_{i-1}}) \rvert$ $\alpha = 1200$	Arithmetic $x_1 = 16000$ $x_i = x_{i-1} + 16000$	Luby 512
Heuristic	VSIDS (3% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)
Polarity	if $\#occ(l) > \#occ(\neg l)$ $l = true$ else $l = false$	Progr. saving	false	Progr. saving
Learning	CDCL (ext. [ABH+08])	CDCL	CDCL	CDCL (ext. [ABH+08])
Cl. sharing	size ≤ 8	size ≤ 8	size ≤ 8	size ≤ 8

Table 4.1: ManySAT: different strategies

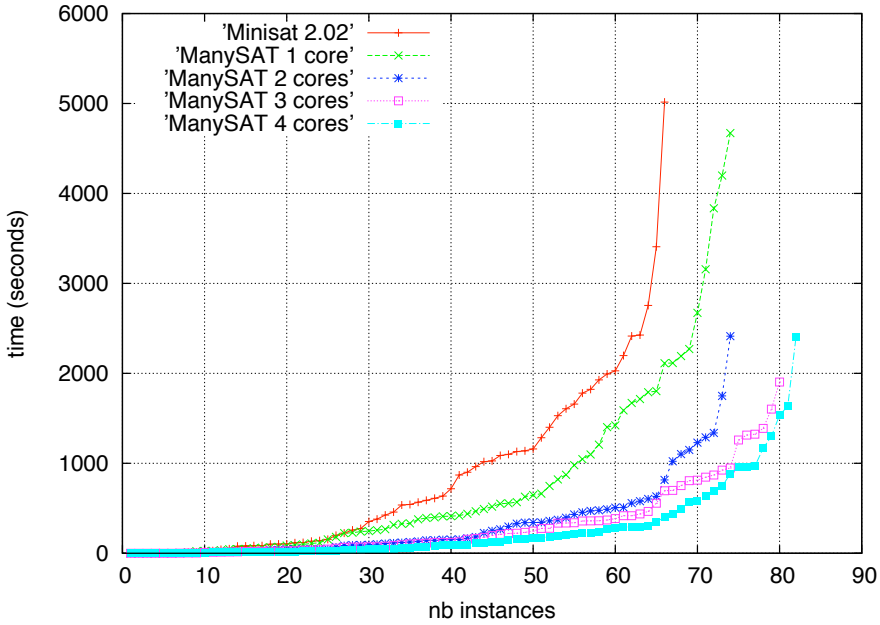


Figure 4.5: SAT-Race 2008: ManySAT e=8, m=1..4 against Minisat 2.02

	ManySAT	**pMinisat**	**MiraXT**
SAT	45	44	43
UNSAT	45	41	30

Table 4.2: SAT-Race 2008: comparative performance (number of problems solved)

The Table 4.2 shows the number of problems (out of 100) solved before the time limit for ManySAT, pMinisat [CS08], and MiraXT [LSB07] - these solvers are described in the next section. We can see that ManySAT solves 5 more problems than pMinisat, which solves 12 more problems than MiraXT. Interestingly, the performance of our method is well balanced between SAT and UNSAT problems. Table 4.3 shows the speed-up provided by these parallel SAT algorithms as opposed to the best sequential algorithm of the Sat-Race 2008, Minisat 2.1. We can see that on average, ManySAT is able to provide a superlinear speed-up of 6.02. It is the only solver able of such performance. The second best provides on average a speed-up of 3.10, far from linear. When we consider the minimal speed-up we can see that the performance of the first two solvers is pretty similar. They decrease the performance against the best sequential solver of the 2008 Sat-Race by up to a factor 4, while the third solver decreases the performance by a factor 25. Finally, the maximal speed-up is given by ManySAT which can

	ManySAT	**pMinisat**	**MiraXT**
Average speed-up	6.02	3.10	1.83
by SAT/UNSAT	8.84 /3.14	4.00/2.18	1.85/1.81
Minimal speed-up	0.25	0.34	0.04
by SAT/UNSAT	0.25/0.76	0.34/0.46	0.04/0.74
Maximal speed-up	250.17	26.47	7.56
by SAT/UNSAT	250.17/4.74	26.47/10.57	7.56/4.26

Table 4.3: SAT-Race 2008: parallel solvers against the best sequential solver (Minisat 2.1)

be up to 250 times faster than Minisat 2.1. These detailed results show that the performance of the parallel solvers is usually better on SAT problems than on UNSAT ones.

	ManySAT	**pMinisat**	**MiraXT**
Average variation	13.7%	14.7%	15.2%
by SAT/UNSAT	22.2%/5.5%	23.1%/5.7%	19.5%/9.7%

Table 4.4: SAT-Race 2008: runtime variation of parallel solvers

It is well known that parallel search is not deterministic. Table 4.4 gives the average runtime variation of each parallel solver. ManySAT exhibits a lower variation than the other techniques, but the small differences between the solvers do not allow us to draw any definitive conclusion.

4.6 Control-based Clause Sharing

The clause sharing approach based on some predefined size limit has several flaws. The first and most apparent being that an overestimated value might induce a very large cooperation overhead, while an underestimated one might completely inhibit the cooperation. The second flaw comes from the observation that the size of learnt clauses tends to increase over time, leading to an eventual halt of the cooperation. The third flaw is related to the internal dynamic of modern solvers which tend to focus on particular subproblems thanks to the activity/restart mechanisms. In parallel SAT, this can lead two search processes toward completely different subproblems where clause sharing becomes pointless.

We propose a dynamic clause sharing policy which uses pairwise size limits to control the exchange between any pair of processing units. Initially, high limits are used to enforce the cooperation, and allow pairwise exchanges. On a regular basis, each unit considers the number of foreign clauses received from other units. If this number is below/above a predefined threshold, the pairwise limits are increased/decreased. This mechanism allows the system to maintain a

throughput. It addresses the flaws one and two. To address the last flaw related to the poor relevance of the shared clauses, we extend our policy to integrate the quality of the exchanges. Each unit evaluates the quality of the received clauses, and the control is able to selectively increase/decrease the pairwise limits based on the underlying quality of the recently communicated clauses. The rationale being that the information recently received from a particular source is qualitatively linked to the information which could be received from it in the very near future. The evolution of the pairwise limits w.r.t., the throughput or quality criterion follows an AIMD (Additive-Increase-Multiplicative-Decrease) feedback control-based algorithm (see section 4.3).

4.6.1 Throughput and Quality-based Control Policies

In this section, we describe our dynamic control-based clause sharing policies which control the exchange between any pair of processing units through dynamic pairwise size limits.

The first policy controls the throughput of clause sharing. Each unit considers the number of foreign clauses received from other units. If this number is below/above a predefined throughput-threshold, the pairwise limits are all increased/decreased using an AIMD feedback algorithm. The second policy is an extension of the previous one. It introduces a measure of the quality of foreign clauses. With this information, the increase/decrease of the pairwise limits become proportional to the underlying quality of the clauses shared by each unit. The first (respectively second) policy allows the system to maintain a throughput (respectively throughput of better quality).

We consider a parallel SAT solver with n different processing units. Each unit u_i corresponds to a SAT solver with clause learning capabilities. Each solver can either work on a subspace of the original instance as in divide-and-conquer techniques, or on the full problem, as in ManySAT. We assume that these different units communicate through a shared memory (as in multicore architectures).

In our control strategy, we consider a control-time sequence as a set of steps t_k with $t_0 = 0$ and $t_k = t_{k-1} + \alpha$ where α is a constant representing the time window defined in term of number of conflicts. The step t_k of a given unit u_i corresponds to the conflict number $k \times \alpha$ encountered by the solver associated to u_i. In the sequel, when there is no ambiguity, we sometimes note t_k simply k. Then, each unit u_i can be defined as a sequence of states $S_i^k = (\mathcal{F}, \Delta_i^k, R_i^k)$, where $\mathcal{F}$ is a CNF formula, Δ_i^k the set of its proper learnt clauses and R_i^k the set of foreign clauses received from the other units between two consecutive steps $k-1$ and k. The different units achieve pairwise exchange using pairwise limits. Between two consecutive steps $k-1$ and k, a given unit u_i receives from all the other remaining units u_j where $0 \leq j < n$ and $j \neq i$ a set of learnt clauses $\Delta_{j\to i}^k$ of length less or equal to a size limit $e_{j\to i}^k$ i.e., $\Delta_{j\to i}^k = \{c \in \Delta_j^k / \ |c| \leq e_{j\to i}^k\}$. Then, the set R_i^k can be formally defined as $\cup_{0\leq j<n, j\neq i} \Delta_{j\to i}^k$.

Using a fixed throughput threshold T of shared clauses, we describe our control-based policies which allow each unit u_i to guide the evolution of the size limit

$e_{j\to i}$ using an AIMD feedback mechanism.

Throughput-based Control

As illustrated in figure 4.6, at step k a given unit u_i checks whether the throughput is exceeded or not. if $|R_i^k| < T$ (respectively $|R_i^k| > T$) the size limit $e_{j\to i}^{k+1}$ is additively increased (respectively multiplicatively decreased). More formally, the upper bound $e_{j\to i}^{k+1}$ on the size of clauses that a solver j shares with the solver i between k and $k+1$ are changed using the following AIMD function:

$aimdT(R_i^k)\{$
$\forall j | 0 \leq j < n, j \neq i$
$e_{j\to i}^{k+1} = \begin{cases} e_{j\to i}^k + \frac{b}{e_{j\to i}^k}, if(|R_i^k| < T) \\ e_{j\to i}^k - a \times e_{j\to i}^k, if(|R_i^k| > T) \end{cases}$ } where a and b are positive constants.

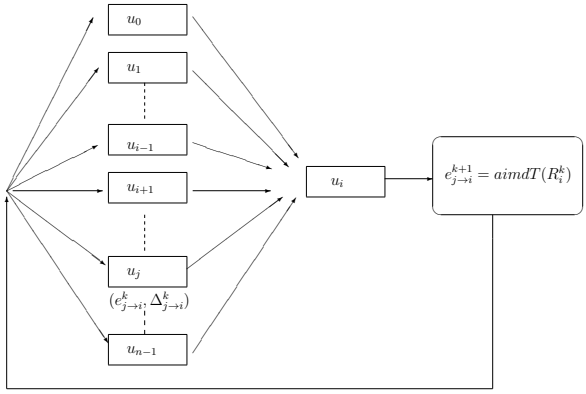


Figure 4.6: Throughput based control policy

Throughput and Quality Based Control

In this policy, to control the throughput of a given unit u_i, we introduce a quality measure $Q_{j\to i}^k$ (see definition 3) to estimate the relative quality of the clauses received by u_i from u_j. In the throughput and quality based control policy, the evolution of the size limit $e_{j\to i}^k$ is related to the estimated quality. Our quality measure is defined using the activity of the variables at the basis of VSIDS heuristic [MMZ+01] another important component of modern SAT solvers. The variables with greatest activity represent those involved in most of the (recent)-conflicts. Indeed, when a conflict occurs, the activity of the variables whose literals appear in the clauses encountered during the generation of a learnt clause are updated. The most active variables are those related to the current part of the search space. Consequently, our quality measure exploits these activities to quantify the relevance of a clause learnt by unit u_j

to the current state of a given unit u_i. To define our quality measure, suppose that, at any time of the search process, we have $\mathcal{A}_i^{max}$ the current maximal activity of u_i's variables, and $\mathcal{A}_i(x)$ the current activity of a given variable x.

Definition 3 (Quality). *Let c be a clause and $\mathcal{L}_{\mathcal{A}_i}(c) = \{x/x \in c \text{ s.t. } \mathcal{A}_i(x) \geq \frac{\mathcal{A}_i^{max}}{2}\}$ the set of active literals of c with respect to unit u_i. We define $\mathcal{P}_{j\to i}^k = \{c/c \in \Delta_{j\to i}^k \text{ s.t. } |\mathcal{L}_{\mathcal{A}_i}(c)| \geq Q\}$ be the set of clauses received by i from j between steps $k-1$ and k with at least Q active literals. We define the quality of clauses sent by u_j to u_i at a given step k as $Q_{j\to i}^k = \frac{|\mathcal{P}_{j\to i}^k|+1}{|\Delta_{j\to i}^k|+1}$*

Our throughput and quality based control policy change the upper bound $e_{j\to i}^{k+1}$ on the size of clauses that a solver j shares with the solver i between k and $k+1$ using the following AIMD function:

$aimdTQ(R_i^k)\{$
$\forall j | 0 \leq j < n, j \neq i$

$$e_{j\to i}^{k+1} = \begin{cases} e_{j\to i}^k + (\frac{Q_{j\to i}^k}{100}) \times \frac{b}{e_{j\to i}^k}, if(|R_i^k| < T) \\ e_{j\to i}^k - (1 - \frac{Q_{j\to i}^k}{100}) \times a \times e_{j\to i}^k, if(|R_i^k| > T) \end{cases}$$

$\}$ where a and b are positive constants.

As shown by the AIMD function of the throughput and quality based control policy, the adjustment of the size limit depends on the quality of shared clauses. Indeed, as it can be seen from the above formula, when the exchange quality between u_j and u_i ($Q_{j\to i}^k$) tends to 100% (respectively 0%), then the increase in the limit size tends to be maximal (respectively minimal) while the decrease tends to be minimal (respectively maximal). Our aim in this second policy is to maintain a throughput of good quality. The rationale being that the information recently received from a particular source is qualitatively linked to the information which could be received from it in the very near future.

4.6.2 Experiments

Our tests were done on Intel Xeon Quadcore machines with 16GB of RAM running at 2.3Ghz. We used a timeout of 1500 seconds for each problem. ManySAT was used with 4 DPLLs strategies each one running on a particular core (unit). To alleviate the effects of unpredictable threads scheduling, each problem was solved three times and the average was taken.
Our dynamic clause sharing policies were added to ManySAT and compared against ManySAT with its default static policy *ManySAT e=8* which exchanges clauses up to size 8. Remark that since each pairwise limit is read by a unit, and updated by another one, our proposal can be integrated without any lock. We have selected $a = 0.125, b = 8$ for aimdT and aimdTQ, associated to a time window of $\alpha = 10000$ conflicts. The throughput T is set to $\frac{\alpha}{2}$ and the upper

bound Q on the number of active literals per clause c is set to $\frac{|c|}{3}$ (see definition 3). Each pairwise limit $e_{j\to i}$ was initialized to 8.
The Table 4.5 presents the results on the 100 industrial problems of the 2008 SAT-Race. The problem set contains families with several instances or individual instances.
From left to right we present, the family/instance name, the number of instances per family. Results associated to the standard ManySAT, with the number of problems solved before timeout, and the associated average runtime. The right part reports results for the two dynamic policies. For each dynamic policy we provide $\bar{e}$, the average of the $e_{j\to i}$ observed during the computation. The last row provides for each method, the total number of problems solved, and the cumulated runtime. For the dynamic policies, it also presents the average of the $\bar{e}$ values.
At that point we have to stress that the static policy ($e = 8$) is optimal in the way that it gives the best average performance on this set of problems. We can observe that the static policy solves 83 problems while the dynamic policies aimdT and aimdTQ solve respectively 86 and 89 problems. Except on the *ibm_** and *manol_** families, the dynamic policies always exhibit a runtime better or equivalent to the static one. Unsurprisingly, when the runtime is significant but does not drastically improve over the static policy, the values of $\bar{e}$ are often close to 8, i.e., equivalent to the static size limit. When we consider the last row, we can see that the aimdT is faster than the aimdTQ. However, this last policy solves more problems. We can explain this as follows. The quality-based policy intensifies the search by favoring the exchange of clauses related to the current exploration of each unit. This intensification leads to the resolution of more difficult problems. However, it increases the runtime on easier instances where a more diversified search is often more beneficial. Overall these results are very good since our dynamic policies are able to outperform the best possible static tuning.

4.7 Summary

We have presented ManySAT, a portfolio-based parallel SAT solver which advantageously exploits multicore architectures. ManySAT is based on an understanding of the main weakness of modern sequential SAT solvers, their sensitivity to parameter tuning and their lack of robustness. As a result, ManySAT uses a portfolio of complementary sequential algorithms, and let them cooperate in order to improve further the overall performance. This design philosophy of ManySAT which clearly contrasts with well known parallel SAT solvers, is directly inspired by our work in the previous M- framework for distributed constraint satisfaction problems. The good performance obtained by ManySAT on industrial SAT instances clearly suggests that the parallel portfolio approach is more interesting than the traditional divide-and-conquer one.
We have also presented how knowledge sharing could be finely controlled through dynamic clause sharing policies which can adjust the size of shared clauses be-

		ManySAT e=8		ManySAT aimdT			ManySAT aimdTQ		
family/instance	#inst	#Solved	time(s)	#Solved	time(s)	$\bar{e}$	#Solved	time(s)	$\bar{e}$
ibm_*	20	19	**204**	19	218	7	19	286	6
manol_*	10	10	**117**	10	**117**	8	10	205	7
mizh_*	10	6	762	7	746	6	**10**	**441**	5
post_*	10	9	325	9	**316**	7	9	375	7
velev_*	10	8	585	8	**448**	5	8	517	7
een_*	5	5	2	5	2	8	5	2	7
simon_*	5	5	111	5	84	10	5	**59**	9
bmc_*	4	4	7	4	7	7	4	**6**	9
gold_*	4	1	1160	1	**1103**	12	1	1159	12
anbul_*	3	2	742	**3**	**211**	11	3	689	11
babic_*	3	3	2	3	2	8	3	2	8
schup_*	3	3	129	3	**120**	5	3	160	5
fuhs_*	2	2	90	2	**59**	11	2	77	10
grieu_*	2	1	783	1	**750**	8	1	**750**	8
narain_*	2	1	786	1	**776**	8	1	792	8
palac_*	2	2	20	2	**8**	3	2	54	7
aloul-chnl11-13	1	0	1500	0	1500	11	0	1500	10
jarvi-eq-atree-9	1	1	70	1	69	25	1	**43**	17
marijn-philips	1	0	1500	**1**	1133	34	1	**1132**	29
maris-s03-gripper11	1	1	11	1	11	10	1	11	8
vange-col-abb313gpia-9-c	1	0	1500	0	1500	12	0	1500	12
Total/(average)	100	83	10406	86	9180	(10.28)	89	9760	(9.61)

Table 4.5: SAT-Race 2008, industrial problems

tween any pair of processing units. The first one controls the overall number of exchanged clauses whereas the second additionally exploits the relevance quality of shared clauses. This part has been described in [HJS09].
As stated here, our four-cores portfolio was carefully crafted in order to mix complementary strategies. If ManySAT could be run on dozens of computing units, what would be the performance? We have considered this question in a more general context in [BHS09]. This work presents the first study on scalability of constraint solving on 100 processors and beyond. It proposes techniques that are simple to apply and shows empirically that they scale surprisingly well. It proves that portfolio-based approaches can also scale-up to several dozens of processors.
Finally, as stated in the introduction, SAT is now applied to other domains. One domain which particularly benefits from the recent advances in SAT is Satisfiability Modulo Theory [NOT06]. There, our ManySAT approach has been integrated to the Z3 SMT solver [dMB08], allowing it to achieve impressive speed-ups on several classes of problems [WHdM09].

In the next chapter we will consider purely sequential search. We will present an approach which can incrementally and dynamically learn the performance of different solver's settings in relation with the actual input distribution.

Chapter 5

Continuous Search

This line of work was initiated in Cambridge, while supervising Frank Hutter in an MSR summer internship. This original work had two positive outcomes. First of all, it boosted Frank interest in automated parameter tuning and helped his decision to choose this domain as a dissertation topic [Hut09]. Second of all, it led two years later to the creation of the *Adaptive Combinatorial Search for e-Science (Adapt)* research project in the MSR-INRIA joint lab. This project co-directed with Marc Schoenauer aims at improving the applicability of constraint-based or heuristic-based solvers to complex scientific problems. The goal in this project is to develop tools able to automatically choose the optimal parameter configuration of a given search algorithm for a given problem or class of problems.

This line of work led to several publications and one US patent:

- Frank Hutter and Youssef Hamadi. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK, jan 2005.
- Frank Hutter, Youssef Hamadi, Holger H. Hoos, and Kevin Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In Frédéric Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2006.
- Alejandro Arbelaez, Youssef Hamadi, and Michele Sebag. Online heuristic selection in constraint programming. In *International Symposium on Combinatorial Search (SoCS)*, Lake Arrowhead, USA, July 2009.
- Alejandro Arbelaez and Youssef Hamadi. Continuous Search in Constraint Programming: An Initial Investigation. In Karen Petrie and Olivia Smith, editors, Constraint Programming Doctoral Program, pages 7-12, Lisbon, Portugal, September 2009.
- Alejandro Arbelaez, Youssef Hamadi, and Michele Sebag. Continuous Search in Constraint Programming. In Eric Gregoire, editor, 22th International Conference on Tools With Artificial Intelligence (ICTAI), volume 1, pages 53-60, Arras, France, October 2010. IEEE.
- US Patent No. 7890439. Tuning of Problem Solvers, Youssef Hamadi, Feb 15, 2011 (filed in 2006).

This Chapter presents some work performed during the co-supervision of the PhD of Alejandro Arbelaez (MSR-INRIA joint-lab).

5.1 Introduction

In the previous chapters, we have seen that portfolios of algorithms can positively impact the robustness of Search. In Chapter 3, our portfolio was using

multiple variables ordering heuristics whose executions were interleaved at the agent level. In Chapter 4, we moved to fully fledged parallelism with a portfolio of parallel CDCL solvers competing and cooperating to tackle a given SAT instance.

The present chapter considers the sequential case where a given Constraint Programming engine is used to successively solve problems coming from a given application domain. The objective is to incrementally learn a predictive model able to accurately match instances features to good solver's parameters. The learning is possible thanks to the relative coherence of the instances, and the goal is to eventually achieve top performance for the underlying application domain.

In Constraint Programming, properly crafting a constraint model which captures all the constraints of a particular problem is often not enough to ensure acceptable runtime performance. Additional tricks, e.g. adding redundant and channeling constraints, or using some global constraint (depending on your constraint solver) which can efficiently do part of the job, are required to achieve efficiency. Such tricks are far from being obvious, unfortunately; they do not change the solution space, and users with a classical mathematical background might find it hard to see why adding redundancy helps.

For this reason, users are often left with the tedious task of tuning the search parameters of their constraint solver, and this again, is both time consuming and not necessarily straightforward. Parameter tuning indeed appears to be conceptually simple (i/ try different parameter settings on representative problem instances, ii/ pick up the setting yielding best average performance). Still, most users would easily consider instances which are not representative of their problem, and get misled.

The goal of the work presented in this chapter is to allow any user to eventually get their constraint solver achieving a top performance on their problems. The proposed approach is based on the original concept of Continuous Search (CS), gradually building a heuristics model tailored to the user's problems, and mapping a problem instance onto some appropriate parameter setting. A main contribution compared to the state of the art (see [SM08] for a recent survey; more in section 5.4) is to relax the requirement of a large set of representative problem instances to be available beforehand to support offline training. The heuristics model is initially empty (set to the initial default parameter setting of the constraint solver) and it is enriched along a lifelong learning approach, exploiting the problem instances submitted by the user to the constraint solver. Formally, CS interleaves two functioning modes. In production or exploitation mode, the instance submitted by the user is processed by the constraint solver; the current heuristics model is used to parameterize the constraint solver depending on the instance at hand. In learning or exploration mode, CS reuses the last submitted instance, running other heuristics than the one used in production mode in order to find which heuristics would have been most efficient for this instance. CS thus gains some expertise relative to this particular instance, which is used to refine the general heuristics model through Machine Learning (section 5.3.2). During the exploration mode, new information is thus generated

and exploited in order to refine the heuristics model, in a transparent manner: without requiring the user's input and by only using the idle computer's CPU cycles.
We claim that the CS methodology is realistic (most computational systems are always on, especially production ones, most systems waste a large amount of CPU cycles) and compliant with real-world settings, where the solver is critically embedded within large and complex applications. The CS computational cost must be balanced against the huge computational cost of offline training [XHHLB07]. Finally, lifelong learning appears a good way to construct an efficient and agnostic heuristics model, and able to adapt to new modelling styles or new classes of problem.

5.2 Related Work

This section briefly reviews and discusses some related works, devoted to heuristic selection within CP and SAT solvers.
SATzilla [XHHLB07] is a well known SAT portfolio solver which is built upon a set of features. Roughly speaking SATzilla includes two kinds of basic features: general features such as number of variables, number of propagators, etc. and local search features which actually probe the search space in order to estimate the difficulty of each problem-instance for a given algorithm. The goal of SATzilla is to learn a run-time prediction function by using a linear regression model. In the same direction of SATzilla in [HW09] Haim et al., build the portfolio taking into account several restarts policies for a set of well known SAT solvers.
CPHydra [OHH$^+$08] is a portfolio approach based on case-based reasoning; it maintains a database with all solved instances (so-called *cases*). Later on, once a new instance I arrives a set of similar cases C is computed and based on C it builds a switching policy selecting a set of CSP solvers that maximizes the possibilities of solving I within a given amount of time.
The approach most similar to the presented one is that of [SM07], who likewise apply Machine Learning techniques to perform on-line combination of heuristics into search tree procedures. Unfortunately, this work requires an important number of training instances to build a model with good generalization property.
In [CB05] low-knowledge is used to select the best algorithm in the context of optimization problems, this work assumes a black-box optimization scenario where the user has no information about the problem or even about the domain of the problem, and the only known information is the output (i.e., solution cost for each algorithm in the portfolio). Unfortunately this mechanism is only applicable to optimization problems and cannot be used to solve CSPs.
The purpose in *The Adaptive Constraint Engine* (ACE) [EFW$^+$02] is to unify the decision of several heuristics in order to guide the search process. In this way, each heuristic votes for a possible variable/value decision to solve a CSP. Afterwards, a global controller selects the most appropriate pair variable/value according to previously (offline) learnt weights associated to each heuristic. The

authors however did not present any experimental scenario taking into account any restart strategy, although these nowadays are an essential part of constraint solvers
Combining Multiple Heuristics Online [SGS07] and Portfolios with deadlines [WB08] are designed to build a scheduler policy in order to switch the execution of *black-box* solvers during the resolution process. However, in these papers the switching mechanics is learnt/defined beforehand, while our approach relies on the use of machine learning to on-the-fly switch the execution of heuristics.
Finally, in [AST09] and [HHS07] the authors studied the automatic configuration problem which objective is to find the best parameters of a given algorithm in order to efficiently solve a class of problems.

5.3 Background and Notations

5.3.1 Constraint Satisfaction Problems

A constraint Satisfaction Problem (*CSP*) is a triple (X, D, C) where, X represents a set of variables, D a set of associated domains (i.e., possible values for the variables) and C a finite set of constraints.
Solving a CSP involves finding a solution, i.e., an assignment of values to variables such as all constraints are satisfied. If a solution exists the problem is stated as satisfiable and unsatisfiable otherwise. A depth-first search backtracking algorithm can be used to tackle CSPs. At each step of the search, an unassigned variable X and a valid value v for X are selected, the exploration of variables/values is combined with a look-ahead strategy able to narrow the domains of the variables and reduce the remaining search space through constraint propagation. Restarting the search engine [GSK98, KHR$^+$02] helps to reduce the effects of early mistakes in the search process. A restart is done when some cutoff limit in the number of failures (backtracks) is met (i.e., at some point in the search tree), before restarting the search each heuristic stores its ranking metrics in order to start the next tree-based search.
In this work, we consider five well known variable selection heuristics. *min-dom* [HE79] selects the variable with the smallest domain, *wdeg* [BHLS04b] selects the variable which is involved in the highest number of failed constraints, *dom-deg* selects the variable which minimizes the ratio $\frac{dom}{deg}$, *dom-wdeg* [BHLS04b] selects the variable which minimizes the ratio $\frac{dom}{wdeg}$ and *impacts* [Ref04] selects the (variable, value) pair which maximizes the reduction of the remaining search space. While only deterministic heuristics will be considered, the proposed approach can be extended to randomized algorithms by following the approach proposed in [HHHLB06].

5.3.2 Supervised Machine Learning

Supervised Machine Learning exploits data labelled by the expert to automatically build hypotheses emulating the expert's decisions [Vap95]. Only the binary

classification case will be considered in the following. Formally, a learning algorithm processes a training set $\mathcal{E} = \{(x_i, y_i), x_i \in \Omega, y_i \in \{1, -1\}, i = 1 \dots n\}$ made of n examples (x_i, y_i), where x_i is the example description (e.g. a vector of values, $\Omega = \mathbb{R}^d$) and y_i is the associated label; example (x, y) is referred to as positive (respectively, negative) iff y is 1 (resp., -1). The learning algorithm outputs a hypothesis $f : \Omega \mapsto Y$ associating to each example description x a label $y = f(x)$ in $\{1, -1\}$. Among ML applications are pattern recognition, ranging from computer vision to fraud detection [LB08], game playing [GS07], or autonomic computing [RBea05].

Among the prominent ML algorithms are *Support Vector Machines* (SVM) [CST00]. Linear SVM considers real-valued positive and negative instances ($\Omega = \mathbb{R}^d$) and constructs the separating hyperplane which maximizes the margin, i.e. the minimal distance between the examples and the separating hyperplane. The margin maximization principle provides good guarantees about the stability of the solution and its convergence towards the optimal solution when the number of examples increases.

The linear SVM hypothesis $f(x)$ can be described from the sum of the scalar products between the current instance x and some of the training instances x_i, called support vectors:

$$f(x) = < w, x > +b = \sum \alpha_i < x_i, x > +b$$

The SVM approach can be extended to non-linear spaces, by mapping the instance space Ω into a more expressive feature space $\Phi(\Omega)$. This mapping is made implicit through the so-called *kernel trick*, by defining $K(x, x') = < \Phi(x), \Phi(x') >$; it preserves all good SVM properties provided the kernel be positive definite. Among the most widely used kernels are the Gaussian kernel ($K(x, x') = exp\{-\frac{||x-x'||^2}{\sigma^2}\}$) and the polynomial kernel ($K(x, x') = (< x, x' > +c)^d$). More complex separating hypotheses can be built on such kernels,

$$f(x) = \sum \alpha_i K(x_i, x) + b$$

using the same learning algorithm core as in the linear case. In all cases, a new instance x is classified as positive (respectively negative) if $f(x)$ is positive (resp. negative).

5.4 Continuous Search in Constraint Programming

The Continuous Search paradigm, illustrated on Figure 5.1, considers a functioning system governed from a heuristics model (which could be expressed as e.g., a set of rules, a knowledge base, a neural net). The core of continuous search is to exploit the problem instances submitted to the system along a 2-step process:

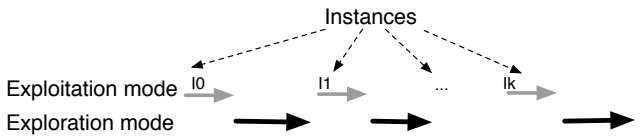


Figure 5.1: Continuous Search scenario

1. Exploitation mode: unseen problem instances are solved using the current heuristics model;
2. Exploration mode:
   (a) these instances are solved with other heuristics, yielding new information. This information associates to the description x of the example (accounting for the problem instance and the heuristics), a boolean label y (the heuristics improves/does not improve on the current heuristics model);
   (b) the training set $\mathcal{E}$, augmented with these new examples (x, y), is used to revise or relearn the heuristics model.

The Exploitation or production mode (step 1) aims at solving new problem instances as quickly as possible. The Exploration or learning mode (steps 2 and 3) aims at learning a more accurate heuristics model.

Definition 1. *A continuous search system is endowed with a heuristics model, which is used as is to solve the current problem instance in production mode, and which is improved using the previously seen instances in learning mode.*

Initially, the heuristics model of a continuous search system is empty, that is, it is set to the default settings of the search system. In the proposed CS-based constraint programming, the default setting is a given heuristics noted *DEF* in the following (section 5.5). Assumedly, *DEF* is a reasonably good strategy on average; the challenge is to improve on *DEF* for the particular types of instances which have been encountered in production mode.

5.5 Dynamic Continuous Search

The Continuous Search paradigm is applied to a restart-based constraint solver, defining the *dyn-CS* algorithm. After a general overview of *dyn-CS*, this section details the different modules thereof.
Figure 5.2 depicts the general scheme of *dyn-CS*. The constraint-based solver involves several restarts of the search. A restart is launched after the number of backtracks in the search tree reaches a user-specified threshold. The search stops after a given time limit. Before starting the tree-based search and after

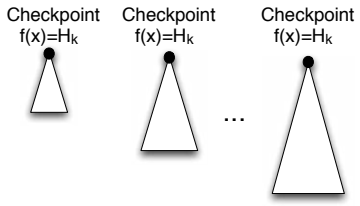


Figure 5.2: *dyn-CS*: selecting the best heuristic at each restart point

each subsequent restarts, the description x of the problem instance is computed (section 5.5.1). We will call checkpoints the calculation of these descriptions.
In production mode, the heuristics model f is used to compute the heuristic $f(x)$ to be applied for the entire checkpoint window, i.e., until the next restart. Not to be confused with the *choice point* which selects a variable/value pair at each node in the search tree, *dyn-CS* selects the most promising heuristic at a given checkpoint and uses it for the whole checkpoint window. In learning mode, other combination of heuristics are applied (section 5.5.4) and the eventual result (depending on whether the other heuristics improved on heuristics $f(x)$) leads to build training examples (section 5.5.3). The augmented training set is used to relearn the heuristics model $f(x)$.

5.5.1 Representing Instances: Feature Definition

At each checkpoint (or restart), the description of the problem instance is computed including static and dynamic features.
While a few of these descriptors had already been used in SAT portfolio solvers [HHHLB06, XHHLB07], many descriptors had to be added as CSPs are more diverse than SAT instances: SAT instances only involve boolean variables and clauses, contrasting with CSPs using variables with large domains, and a variety of constraints and pruning rules.

Static Features

Encode the general description of a given problem instance; they are computed once for each instance as they are not modified along the resolution process. The static features also allow one to discriminate between types of problems, and different instances.

- **Problem definition** (4 features): Number of variables, constraints, variables assigned/not assigned at the beginning of the search.

- **Variables size information** (6 features): `Size prod`, `sum`, `min`, `max`, `mean` and `variance` of all variables domain size.

- **Variables degree information** (8 features): `min`, `max`, `mean` and `variance` of all variables degree (resp. variables' domain/degree)
- **Constraints Information** (6 features): The degree (or arity) of a given constraint c is represented by the total number of variables involved in c. Likewise the size of c is represented by the product of its corresponding variables domain sizes. Taking into account this information, the following features are computed `min`, `max`, `mean` of constraints size and degree.
- **Filtering cost category** (8 features): Each constraint c is associated a category[1]. In this way, we compute the number of constraints for each category. Intuitively each category represents the implementation cost of the filtering algorithm. $Cat = \{$*Exponential*, *Cubic*, *Quadratic*, *Linear expensive*, *Linear cheap*, *Ternary*, *Binary*, *Unary*$\}$. Where *Linear expensive* (resp. *cheap*) indicates the complexity of a linear equation constrain and the last three categories indicate the number of variables involved in the constraint. More information about the filtering cost category can be found in [Gec06].

Dynamic Features

Two kinds of dynamic features are used to monitor the performance of the search effort at a given checkpoint: global statistics describe the progress of the overall search process; local statistics check the evolution of a given strategy.

- **Heuristic criteria** (15 features): each heuristic criteria (e.g., *wdeg*, *dom-wdeg*, *impacts*) is computed for each variable; their `prod`, `min`, `max`, `mean` and `variance` over all variables are used as features.
- **Constraints weight** (12 features): likewise report the `min`, `max`, `mean` and `variance` of all constraints weight (i.e., constraints *wdeg*). Additionally the `mean` for each filtering cost category is used as feature.
- **Constraints information** (3 features): `min`, `max` and `mean` of constraint's *run-prop*, where *run-prop* indicates the number of times the propagation engine has called the filtering algorithm of a given constraint.
- **Checkpoint information** (33 features): for every checkpoint$_i$ relevant information from the previous checkpoint$_{i-1}$ (when available) is included into the feature vector. From checkpoint$_{i-1}$ we include the total number of nodes and maximum search depth. From the latest non-failed node, we consider the total number of assigned variables, satisfied constraints, sum of variables *wdeg* (resp. size and degree) and product of variables degree (resp. *domain*, *wdeg* and *impacts*) of non assigned variables. Finally using the previous 11 features the `mean` and `variance` is computed taking into account all visited checkpoints.

[1] Out of 8 categories, detailed in
http://www.gecode.org/doc-latest/reference/classGecode_1_1PropCost.html

The attributes listed above include a collection of 95 features.

5.5.2 Feature pre-Processing

Feature pre-processing is a most important step in Machine Learning [WF05], which can significantly improve the predictive accuracy of the learned hypothesis. Typically, the descriptive features detailed above are on different scales; the number of variables and/or constraints can be high while the Impact of (variable, value) is between 0 and 1. A data normalization step, scaling down feature values in $[-1, 1]$ (*minmax-normalization*) is used.
Although selecting the most informative features might improve the performance, in this work we do not consider any feature selection algorithm, and only features that are constant over all examples are removed as they offer no discriminant information.

5.5.3 Learning and Using the Heuristics Model

The selection of the best heuristic for a given problem instance is formulated as a binary classification problem, as follows. Let $\mathcal{H}$ denote the set of k candidate heuristics, two particular elements in $\mathcal{H}$ being *DEF* (the default heuristics yielding reasonably good results on average) and *dyn-CS*, the (dynamic) ML-based heuristics model initially set to *DEF*.

Definition 2. *Each training example $p_i = (x_i, y_i)$ is generated by applying some heuristics h ($h \in \mathcal{H}, h \neq$ dyn-CS) at some checkpoint in the search tree of a given problem instance. Description x_i ($\in \mathbb{R}^{97}$) is made of the static feature values describing the problem instance, the dynamic feature values computed at this check point and describing the current search state, and two additional features: checkpoint-id gives the number of checkpoints up to now and cutoff-information gives the cutoff limit of the next restart. The associated label y_i is positive iff the associated runtime (using heuristic h instead of dyn-CS at the current checkpoint) improves on the heuristics model-based runtime (using dyn-CS at every checkpoint); otherwise, label y_i is negative.*

If the problem instance cannot be solved (whatever the heuristics used, i.e., time out during the exploration and exploitation modes), it is discarded (since the associate training examples do not provide any relevant information).
In production mode, the hypothesis f learned from the above training examples (their generation is detailed in next subsection) is used as follows:

Definition 3. *At each checkpoint, for each $h \in \mathcal{H}$, the description x_h and the associated value $f(x_h)$ are computed. If there exists a single h such that $f(x_h)$ is positive, it is selected and used in the subsequent search effort. If there exists several heuristics with positive $f(x_h)$, the one with maximal value is selected*[2]*. If $f(x_h)$ is negative for all h, the default heuristic* DEF *is selected.*

[2]The rationale for this decision is that the margin, i.e. the distance of the example w.r.t the separating hyperplane, is interpreted as the confidence of the prediction [Vap95].

5.5.4 Generating Examples in Exploration Mode

The Continuous Search paradigm uses the idle computer's CPU cycles to explore different heuristic combinations on the last seen problem instance, and see whether one could have done better than the current heuristics model on this instance. The rationale for this exploration is that improving on the last seen instance (albeit meaningless from a production viewpoint since the user already got a solution) will deliver useful indications as to how to best deal with further similar instances. In this way, the heuristics model will expectedly be tailored to the distribution of problem instances actually dealt with by the user.
The CS exploration proceeds by slightly perturbing the heuristics model. Let *dyn-CS* $^{-i,h}$ denote the policy defined as: use heuristics model *dyn-CS* at all checkpoints except the i-th one, and use heuristic h at the i-checkpoint.

Algorithm 1 Exploration-time(instance: $\mathcal{I}$)

1: $\mathcal{E} = \{\}$ //initialize the training set
2: **for all** i in checkpoints($\mathcal{I}$) **do**
3: **for all** h in $\mathcal{H}$ **do**
4: Compute x describing the current checkpoint i and heuristic h
5: **if** $h \neq$ *dyn-CS* **then**
6: Launch *dyn-CS* $^{-i,h}$
7: Define $y = 1$ iff *dyn-CS* $^{-i,h}$ improves on *dyn-CS* and -1 otherwise
8: $\mathcal{E} \leftarrow \mathcal{E} \cup \{x, y\}$
9: **end if**
10: **end for**
11: **end for**
12: return $\mathcal{E}$

Algorithm 1 describes the proposed Exploration mode for Continuous Search. A limited number (10) of checkpoints in the *dyn-CS* based resolution of instance $\mathcal{I}$ are considered (line 2); for each checkpoint and each heuristic h (distinct from the *dyn-CS*), a lesion study is conducted, applying h instead of *dyn-CS* at the i-th checkpoint (heuristics model *dyn-CS* $^{-i,h}$); the example (described from the i-th checkpoint and h) is labelled positive iff *dyn-CS* $^{-i,h}$ improves on *dyn-CS*, and added to the training set $\mathcal{E}$, once the exploration mode for a given instance is finished the hypothesis model is updated by retraining the SVM including the feature pre-processing as stated in section 5.5.2.

5.5.5 Imbalanced Examples

It is well known that one of the heuristics often performs much better than the others for a particular distribution of problems [CB08]. Accordingly, negative training examples considerably outnumber the positive ones (it is difficult to improve on the winning heuristics). This phenomenon, known as *Imbalanced distribution*, might severely hinder the SVM algorithm [AKJ04]. Two simple ways of enforcing a balanced distribution in such cases, intensively examined in the

literature and considered in earlier work [AHS09], are to over-sample examples in the minority class (generating additional positive examples by Gaussianly perturbing the available ones) and/or undersample examples in the majority class.
Another option is to use prior knowledge to rebalance the training distribution. Formally, instead of labeling an example positive (resp, negative) iff the associated runtime is strictly less (resp. greater) than that of the heuristic model, we consider the difference between the runtimes. If the difference is less than some tolerance value dt, then the example is relabeled as positive.
The number of positive examples and hence the coverage of the learned heuristics model increase with dt; in the experiments (Section 5.6), dt is set to 1 minute iff *time-exploitation* (time required to solve a given instance in production mode) is greater than 1 minute, otherwise dt is set to *time-exploitation*.

5.6 Experimental Validation

This section reports on the experimental validation of the proposed Continuous Search approach. All tests were conducted on Linux Mandriva-2009 boxes with 8 GB of RAM and 2.33 Ghz Intel processors.

5.6.1 Experimental Settings

The presented experiments consider 496 CSP instances taken from different repositories.

- **nsp**: 100 *nurse-scheduling* instances from the MiniZinc[3] repository.
- **bibd**: 83 *Balance Incomplete Block Design* instances from the XCSP[4] repository, translated into Gecode using Tailor[5]
- **js**: 130 *Job Shop* instances from the XCSP repository.
- **geom**: 100 *Geometric* instances from the XCSP repository.
- **lfn**: 83 *Langford-number* instances, translated into Gecode using global and channelling constraints.

The learning algorithm used in the experimental validation of the proposed approach is a Support Vector Machine with Gaussian kernel, using the libSVM implementation with default parameters[6]. All considered CSP heuristics (Section 5.3) are home-made implementations integrated in the Gecode 2.1.1 [Gec06] constraint solver. Our *dyn-CS* technique was used as a heuristics model on top

[3] http://www.g12.cs.mu.oz.au/minizinc/download.html
[4] http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html
[5] http://www.cs.st-andrews.ac.uk/~andrea/tailor/
[6] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

of the heuristics set $\mathcal{H} = \{$*dom-wdeg*, *wdeg*, *dom-deg*, *min-dom*, *impacts* $\}$, taking *min-value* as value selection heuristic. The cutoff value used to restart the search was initially set to 1000 and the cutoff increase policy to $\times 1.5$, the same cutoff policy is used in all the experimental scenarios.

Continuous Search was assessed comparatively to the best two dynamic variable ordering heuristics on the considered problems, namely *dom-wdeg* and *wdeg*. It must be noted that Continuous Search, being a lifelong learning system, will depend on the curriculum, that is the order of the submitted instances. If the user "pedagogically" starts by submitting informative instances first, the performance in the first stages will be better than if untypical and awkward instances are considered first. For the sake of fairness, the performance reported for Continuous Search on each problem instance is the median performance over 10 random orderings of the CSP instances.

5.6.2 Practical Performances

Figure 5.3 highlights the Continuous Search results on Langford-number problems, comparatively to *dom-wdeg* and *wdeg*. The x-axis gives the number of problems solved and the y-axis presents the cumulated runtime. The (median) *dyn-CS* performance (grey line) is satisfactory as it solves 12 more instances than *dom-wdeg* (black line) and *wdeg* (light gray line). The dispersion of the *dyn-CS* results depending on the instance ordering is depicted from the set of dashed lines. Let us remark that traditional portfolio approaches such as [HHHLB06, SM07, XHHLB07] do not present such performance variations as they assume a complete set of training examples to be available beforehand.

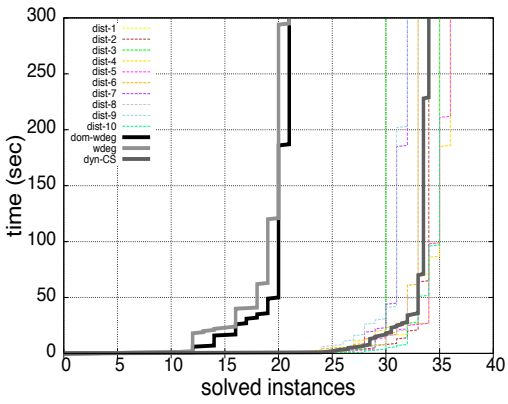
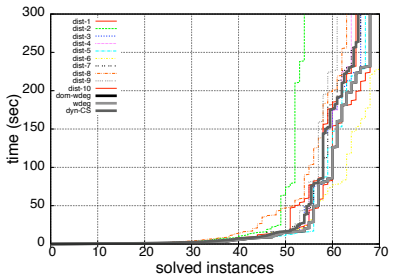
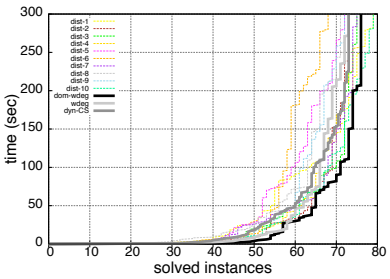


Figure 5.3: Langford-number (lfn): Number of instances solved in less than 5 min with *dyn-CS*, *wdeg*, and *dom-wdeg*. Dashed lines illustrate the performance of *dyn-CS* for a particular instance ordering.

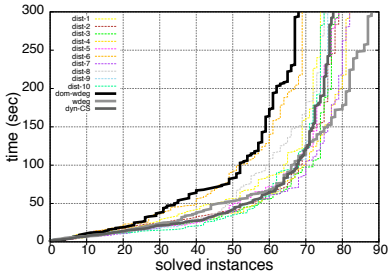
Figure 5.4 depicts the performance of *dyn-CS*, *dom-wdeg* and *wdeg* on all other problem families, respectively (bibd, js, nsp, and geom). On the bibd (Figure 5.4(a)) and js (Figure 5.4(b)) problems, the best heuristics is *dom-wdeg*, solving 3 more instances than *dyn-CS*. Note that *dom-wdeg* and *wdeg* coincide on bibd since all decision variables are boolean.



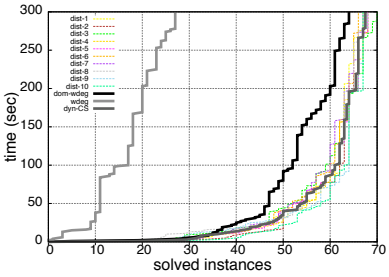
(a) Balance incomplete block designs (bibd)



(b) Job-shop (js)



(c) Nurse scheduling problem (nsp)



(d) Geometric (geom)

Figure 5.4: Number of instances solved in less than 5 minutes

On nsp (Figure 5.4(c)), *dyn-CS* solves 9 more problems than *dom-wdeg*, but is outperformed by *wdeg* by 11 problems. On geom (Figure 5.4(d)), *dyn-CS* improves on the other heuristics, solving respectively 3 more instances and 40 more instances than *dom-wdeg* and *wdeg*.

These results suggest that *dyn-CS* is most often able to pick up the best heuristics on a given problem family, and sometimes able to significantly improve on the best of the available heuristics.

All experimental results are summarized in Table 5.1, reporting for each considered heuristics the number of instances solved (#sol), the total computational cost for all instances (time, in hour), and the average time (avg-time, in minutes) per instance, over all problem families. These results confirm that *dyn-CS* outperforms *dom-wdeg* and *wdeg*, solving respectively 18 and 41 instances more out of 315. Furthermore, it shows that *dyn-CS* is slightly faster than the other

heuristics, with an average time of 2.11 minutes, against respectively 2.39 for *dom-wdeg* and 2.61 for *wdeg*. It is also worth mentioning that the total CPU time required to complete the exploration (or learning) mode after solving a given instance was on average no longer than 2 hours.
Additionally, a random heuristic selection scenario was also experimented (i.e., executing 10 times each instance with a uniform heuristic selection and reporting the median value over the 10 runs). The random selection strategy was able to solve 278 out of 496 instances, 19 instances less than *dom-wdeg* and 37 instances less than *dyn-CS*.
Another interesting lesson learned from the experiments concerns the difficulty of the underlying learning problem, and the generalization error of the learned hypothesis. The generalization error in the Continuous Search framework is estimated by 10-fold Cross Validation on the whole training set (including all training examples generated in exploration mode). Table 5.2 reports on the predictive accuracy of the SVM algorithm (with same default setting) on all problem families, with an average accuracy of 67%. As could have been expected, the predictive accuracy is correlated to the performance of Continuous Search: the problems with best accuracy and best performance improvement are geom and lfn.
To give an order of idea, 62% predictive accuracy was reported in the context of SATzilla [XHHLB07], aimed at selecting of the best heuristic in a portfolio. A direct comparison of the predictive accuracy might however be biased. On the one hand SATzilla errors are attributed to the selection of some near-optimal heuristics, after the authors; on the other hand, Continuous Search would involve several selection steps (in each checkpoint) and could thus compensate from earlier errors.

5.6.3 The Power of Adaptation

Our second experimental test combines instances from different domains in order to show how CS is able to adapt to changing problems distribution. Indeed, unlike classical portfolio-based approaches which can only be applied if the training and exploitation sets come from the same domain, CS can adapt to changes and provide top performances even if the problems change.
In this context, Table 5.3 reports the results on the geom (left) and bibd (right) problems by considering the following two scenarios. In the first scenario, we are going to emulate a portfolio-based search which would use the wrong domain to train. In *nsp-geom*‡, CS incrementally learns while solving the 100 nsp instances, and then solves one by one the 100 geom instances. However, when switching to this second domain, incremental learning is switched off, and checkpoints adaptation uses the model learnt on nsp. In the second scenario, *nsp-geom*† we solve nsp, then geom instances one by one, but this time, we keep the incremental learning on when switching from the first domain to the second one - as if CS was not aware of the transition.
As we can see in the first line of the Table, training on the wrong domain gives poor performance (55 instances solved in 4.1 hours). At contrary, the second

Table 5.1: Total solved instances

Problem	*dom-wdeg*			*wdeg*			*dyn-CS*		
	#sol	time(h)	avg-time(m)	#sol	time(h)	avg-time(m)	#sol	time(h)	avg-time(m)
nsp	68	3.9	2.34	**88**	**2.6**	**1.56**	77	2.9	1.74
bibd	**68**	**1.8**	**1.37**	68	1.8	1.37	65	2.0	1.44
js	**76**	**4.9**	**2.26**	73	5.1	2.35	73	5.2	2.4
lfn	21	5.2	3.75	21	5.3	3.83	**33**	**4.1**	**2.96**
geom	64	3.9	2.34	27	6.8	4.08	**67**	**3.3**	**1.98**
Total	297	19.7	2.39	274	21.6	2.61	**315**	**17.5**	**2.11**

Table 5.2: Predictive accuracy of the heuristics model (10-fold cross validation)

bibd	nsp	geom	js	lfn
63.2%	58.8%	76.9%	63.6%	73.8%

line shows that CS can recover from training on the wrong domain thanks to its incremental adaptation (solving 67 instances in 3.4 hours). The right part of the Table reports similar results for the bibd problem.
As can be observed in nsp-geom† and lfn-bibd†, CS successfully identifies the new distribution of problems solving respectively the same number and 2 less instances than geom and bibd when CS is only applied to this domain starting from scratch. However the detection of the new distribution introduces an overhead in the solving time (see results for single domain in Table 5.1).

Problem	#Sol	time (h)	Problem	#Sol	time (h)
nsp-geom‡	55	4.1	lfn-bibd‡	23	5.3
nsp-geom†	67	3.4	lfn-bibd†	63	2.3

Table 5.3: Total solved instances

5.7 Summary

The main contribution of the presented approach, the Continuous Search framework aims at designing a heuristics model tailored to the user problem distribution, allowing her to get top performance from the constraint solver. The representative instances needed to train a good heuristics model are not assumed to be available beforehand; they are gradually built and exploited to improve the current heuristics model, by stealing the idle CPU cycles of the computing system. Metaphorically speaking, the constraint solver uses its spare time to play against itself and gradually improve its strategy along time; further, this expertise is relevant to the real-world problems considered by the user, all the more so as it directly relates to the problem instances submitted to the system.
The experimental results suggest that Continuous Search is able to pick up the best of a set of heuristics on a diverse set of problems, by exploiting the incoming instances; in 2 out of 5 problems, Continuous Search swiftly builds up a mixed strategy, significantly overcoming all baseline heuristics. With the other classes of problems, its performance is comparable to the best two single heuristics. Our experiments also showed the capacity of adaptation of CS. Moving from one problem domain to another one is possible thanks to its incremental learning capacity. This capacity is a major improvement against classical portfolio-based approaches which only work when offline training and exploitation use instances from the same domain.

Chapter 6

Autonomous Search

The work presented in this Chapter led to the following events and publications:

- International Workshop on Autonomous Search, co-located with CP'07, Providence (RI), USA.
- Guest co-Editor Special Issue on Autonomous Search, Y. Hamadi, E. Montfroy, and F. Saubion, Constraint Programming Letters (CPL), Volume 4, 2008.
- Book chapter, "Autonomous Search", in "CPAIOR 10th anniversary", Y. Hamadi, E. Monfroy, and F. Saubion, Eds. M. Milano and P. Van Hentenryck, Springer 2010.
- Edited book "Autonomous Search", Springer, ISBN 978-3-642-21433-2, January 31, 2012.

6.1 Introduction

The selection and the correct setting of the most suitable algorithm for solving a given problem has already been investigated many years ago [Ric75]. The proposed abstract model suggested to extract features in order to characterize the problem, to search for a suitable algorithm in the space of available algorithms and then to evaluate its performances with respect to a set of measures. These considerations are still valid and this general problem can indeed be considered at least from two complementary points of view:

- Selecting solving techniques or algorithms from a set of available techniques
- Tuning an algorithm with respect to a given instance of a problem

To address these issues, the proposed approaches include tools from different computer science areas, especially from machine learning. Moreover, they have been developed to answer the algorithm selection problem in various fields as described in the recent survey of K. Smith-Miles [SM08].
In this chapter, we will focus on the restriction of this general question to constraint satisfaction and optimization problems. In this particular area, the problem of finding the best configuration in a search space of heuristic algorithms is also related to the recent notion of Hyper-heuristics [BHK+09, BGJ+03, CKS02]. Hyper-heuristics are methods that aim at automating the process of selecting, combining, generating, or adapting several simpler heuristics (or components of such heuristics) to efficiently solve computational search problems. Hyper-heuristics are also defined as "heuristics to choose heuristics" [CS00] or "heuristics to generate heuristics" [BEDP08]. This idea was pioneered in the early 60's with the combination of scheduling rules [FT63, CGTT63]. Hyper-heuristics that manage a set of given available basic search heuristics by means of search strategies or other parameters have been widely used for solving combinatorial problems (see Burke et al. [BHK+09] for a recent survey).

From a practical point of view, Burke et al. [BHK+10] proposed a comprehensive classification of hyper-heuristics considering two dimensions: the nature of the heuristics and the source of the feedback for learning. They thus distinguish between heuristics that select heuristics from a pre-existing set of search heuristics and heuristics that generate new heuristics from basic components. Concerning the feedback, they identify three categories: online learning, offline learning, and no learning. The distinction between online and offline processes was previously proposed in order to classify parameter setting in evolutionary algorithms [EHM99], distinguishing parameter tuning (offline) from parameter control (online).
As classical offline mechanisms, we may mention *portfolio* algorithms [HHHLB06, XHHLB08], where previously acquired knowledge is used in order to select the suitable solving method with regards to a given problem instance. M. Gagliolo et al. [GS08] use reinforcement learning based techniques for algorithm selection.
Online control of heuristics has been widely addressed, for instance in adaptive strategies in evolutionary computation [Thi07, Kra08], in adaptive neighborhood selection for local search [HR06, CB01, PR08] or in constraint programming solvers [EFW+02].
When considering parameter setting, the space of possible algorithms is the set of possible configurations of a given algorithmic scheme induced by the possible values of its parameters that control its computational behavior. Parameter tuning of evolutionary algorithms has been investigated for many years (we refer the reader to the book [LLM07] for a recent survey). Adaptive control strategies were also proposed for other solving approaches such as local search [Hoo02, PK01]. Offline mechanisms are also available for tuning parameters, such as the work of Hutter et al. [HHS07], which proposes to use a local search algorithm in order to automatically find a good (i.e., efficient) configuration of an algorithm in the parameters space. Including this work, a more complete view of the configuration of search algorithms is presented in the PhD thesis of F. Hutter [Hut09]. Revac [NE07, NE06] is a method that uses information theory to identify the most important parameters and calibrate them efficiently. We may also mention that racing techniques [BSPV02, YG04, YG05, YG07] can be used to choose suitable parameters settings when facing multiple choices.
Another important research community that focuses on very related problems has been established under the name *Reactive Search* by R. Battiti et al. [BBM08, BB09]. After focusing on local search with the seminal works on reactive tabu [BT94] or adaptive simulated annealing [Ing89], this community is now growing through the dedicated *Learning and Intelligent OptimizatioN (LION)* conference.

It clearly appears that these approaches share common principles and purposes and have been developed in parallel in different but connected communities. Their foundations rely on the fact that, since the solving techniques and search heuristics are more and more sophisticated and the problems structures more and more intricate, the choice and the correct setting of a solving algorithm is

becoming an intractable task for most users. Therefore, there is a rising need for an alternative problem solving framework. According to the above brief historical review, we have remarked that these approaches have indeed their own specificities that are induced by their seminal supporting works. In this chapter, we propose to integrate the main motivations and goal into the more general concept of Autonomous Search (AS) [HMS08a, HMS08b].
This chapter is organized as follows. In Section 6.2, we describe the general architecture of modern solvers. We present the specificities of autonomous solvers and formalize their solving mechanisms with a set of rules in section . In Section 6.4, we illustrate different solver architectures by providing examples from the literature and we characterize these solvers using our previous rule-based description framework.

6.2 Solvers Architecture

In this section, we present the general basic concepts related to the notion of solver in the context of general constraint problems solving, which provides a general introduction on problem solving. By general problems, we mean optimization or constraint satisfaction problems, whose variables may take their values over various domains (Boolean, integers, real numbers,...). In fact, solving such problems is the main interest of different but complementary communities in computer science: operation research, global optimization, mathematical programming, constraint programming, artificial intelligence, ... Among the different underlying paradigms that are associated to these research areas, we may try to identify common principles, which are shared by the resulting solving algorithms and techniques that can be used for the ultimate solving purpose.

As it has finally be suggested by the notion of metaheuristics [GK03], solvers could be viewed as a general skeleton whose components are selected according to the problem or the class of problems to be solved. Indeed, from our point of view we want to look carefully at the components of the solver that define its structural properties and at its parameters or external features that define its behavior. On one hand, one has to choose the components of the solver and on the other hand one should configure how these internal components are used during the solving process. We identify the core of the solver which is composed by one or several solving algorithms. Note that here we distinguish between the solver and the solving algorithm, which is a part of the solver but corresponds to the real operational solving process. A basic solving algorithm corresponds to the management of solving techniques, abstracted by the notion of operators, making use of a solving strategy that schedules the use of these operators. A solving algorithm is designed of course according to the internal model, that defines the search space, and uses a function to evaluate the elements of the search spaces. All these components can be subjected to various parameters that define their behavior. A given parametrization defines thus what we call a configuration of the solver. At this level, a control layer can be introduced,

especially in an autonomous solver, to manage the previous components and modify the configuration of the solver during the solving process. The general description of a solver architecture is illustrated by Figure 6.1.

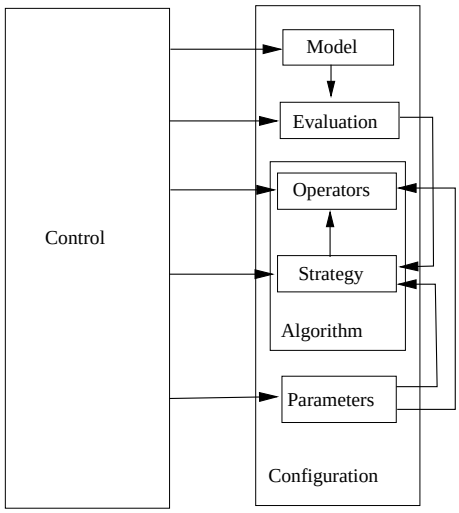


Figure 6.1: The general architecture of a solver

6.2.1 Problem Modeling/Encoding

The encoding of the problem is considered apart from the solver itself. In fact, most of the time, a solver is designed for a specific encoding framework that induces a specific internal representation that corresponds to the model. While the classic CSP modeling framework [Tsa93] is commonly used as a description tool for all solving methods, the internal encoding of the problem and its possible configurations involve different representations (e.g., complete vs. partial assignments, ...). One should note that different modeling and encoding paradigms can be used. In constraint programming [Apt03, Dec03, MS98, Hen89] one could encode constraints as tuples of allowed values or using a more declarative first order language with relations and functions. Moreover, other paradigms can be used to encode CSPs, such as SAT [BHvMW09], and various transformation schemes have been investigated [BHZ06, Wal00, Hoo99]. On the metaheuristics side, the encoding of the possible configurations of the problem has a direct impact on the search space and on the search landscape. For instance, one may include directly some of the constraints of the problem in the encoding as this is the case when using permutations for the Traveling Salesman Problem (TSP [ABCC07]), which corresponds to the constraint: *Each city is visited once and only once*. In genetic algorithms [De 06, ES03b, Mic92] or local search [AL03, HM05], encoding may have a significant impact on the performance of

the algorithm. The encoding of continuous optimization problems (i.e., over real numbers) also requires providing suitable data structures, for instance, floating point representation for genetic algorithms [JM91] or continuous and interval arithmetic in constraint programming [BG06]. The internal representation of the model can be considered as a component of the solver. This representation has of course a direct computational impact on the evaluation function and also on the solving techniques that are implemented through operators.

6.2.2 The Evaluation Function

The evaluation function is related to the nature of the problem. From a general point of view, a function is needed to evaluate possible configurations of the problem with regards to its constraints and variables values. An evaluation function may evaluate the number of conflicts or check the satisfiability of a given constraint set, or use particular consistency notions (global or local). Such a function can also be used to prune the search space when dealing with optimization problems. Again, this notion is more classically used in the context of metaheuristics than in classic complete constraint programming solvers. But it seems rather intuitive to have such a function to assess the current search state in order to be able to check if the solver has reached a solution or not. Moreover, this evaluation function clearly appears when dealing with constraint optimization problems and using branch-and-bound algorithms.

6.2.3 The Solving Algorithm

Our purpose is to distinguish between the basic structure of the algorithm and its configurable components. For instance, in a classic complete constraint solver, the skeleton of the algorithm is the basic backtracking process, whose heuristics and propagation rules can be configured. In an evolutionary algorithm, the core of the solver is constituted by the population management. A solver may include the following components that we have to take into account:

- **A set of operators**: operators are used in the solving process to compute search states. These operators may basically achieve variable instantiation, constraint propagation, local moves, recombination or mutation operators, selection, etc. Most of the time, they are parametrized and use an evaluation function to compute their results (e.g., number of violated constraints or evaluation of the neighborhood in local search algorithms). Note that these operators may be used to achieve a complete search (i.e., able to find a solution or prove unsatisfiability of the problem) or to perform an incomplete search (i.e., find a solution if possible or a sub-optimal solution).

  - Concerning tree search based methods, the notion of operator for performing solving steps during the search process rather corresponds to

basic solving techniques. For instance if we consider a classic backtracking based solver in constraint programming, we need an enumeration operator that is used to assign values to variables and reduction operators that enforce consistencies in order to reduce the domains of the variables. The search process then corresponds to the progressive construction of a search tree whose nodes are subjected to the application of the previously described operators. When considering numerical variables over intervals, we may add splitting operators. Of course these operators may include heuristics concerning the choice of the variables to be enumerated, the choice of the values but also other parameters to adjust their behavior. Indeed, constraint propagation can be formalized by means of rules [Apt03, FA03], which support operators-based description and provide a theoretical framework to assess properties of the solver such as termination.

- On the metaheuristics side, in evolutionary computing [Gol89, De 06, ES03b] we usually consider variation operators (mutation operators and recombination operators) and selection operators. Considering an evolutionary algorithm, it is possible to established some convergence properties such as the famous schemata theorem [Hol75]. There exist some general purpose operators as, for instance, the uniform crossover [Syw89] or the Gaussian mutation [Kje91]. To get better performances, these operators are often designed with respect to the specificities of the problem to be solved. In local search[AL03], local moves are based on neighborhoods functions.

All these operators are most of the time subjected to parameters that may modify their behavior but, more important that also control their application along the search process.

- **A solving strategy**: the solving strategy schedules how operators are used. Back to previous example, in a complete tree-based search process, the strategy will consist in alternating enumeration and constraint propagation. The strategy can be subjected to parameters that will indicate which operators to choose in the general scheduling of the basic solving process.

6.2.4 Configuration of the solver: the Parameters

The solver usually includes parameters that are used to modify the behavior of its components. A configuration of the solver is then an instance of the parameters together with its components. Parameters are variables that can be used in the general search process to decide how the other components are used. These parameters may correspond to various data that will be involved in the choice of the operator to be applied at a given search state. For instance, we may consider the probability of application of the operators (e.g., genetic operators in evolutionary algorithms, the noise in random walk for local search

algorithms [SKC94]) or to some tuning of the heuristics themselves (e.g., tabu list length in Tabu Search [GL97]).
Parameters setting is an important issue for evolutionary algorithms [LLM07]. Parameters setting for local search algorithms is also handled in [BBM08]. In constraint programming much work has been done to study basic choice heuristics (see [EFW+02] for instance), but also to evaluate the possible difficulties related to the classic use of basic heuristics such as heavy-tailed problems [GSCK00] (these studies particularly demonstrate the benefit of randomization when solving multiple instances of a given family of problem compared to the use of a single predefined heuristics) .

6.2.5 Control

Modern solvers also include external or internal mechanisms that allow the solver to change its configuration by selecting the suitable operators to apply, or tuning the parameters, or adding specific information to the model. These mechanisms often include machine learning techniques and will be detailed later. Of course, control rules will often focus on the management of the parameters and/or of the operators of the solver.

6.2.6 Existing Classifications and Taxonomies

As mentioned before, we may identify at least three important domains where related work has already been conducted. These lines of work have led to the use of different terminologies and concepts that we try to recall here.

In evolutionary computing, parameters setting [LLM07] constitutes a major issue and we may recall the taxonomy proposed by Eiben et al. [EHM99] (see Figure 6.2).

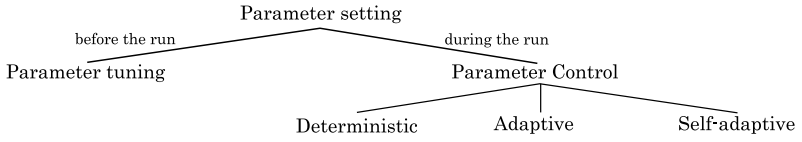


Figure 6.2: Control taxonomy proposed by Eiben et al. [EHM99]

Methods are classified depending on whether they attempt to set parameters before the run (tuning) or during the run (control). The goal of parameter tuning is to obtain parameters values that could be useful over a wide range of problems. Such results require a large number of experimental evaluations and are generally based on empirical observations. Parameter control is divided into three branches according to the degree of autonomy of the strategies. Control is deterministic when parameters are changed according to a previously established schedule, adaptive when parameters are modified according to rules that take

into account the state of the search, and self-adaptive when parameters are encoded into individuals in order to evolve conjointly with the other variables of the problem.
In [SE09], Eiben and Smit recall the difference between numeric and symbolic parameters. In [NSE08], symbolic parameters are called components whose elements are operators. In this chapter, we choose to use the notions of parameters for numeric parameters. As defined above, the operators are configurable components of the solver that implement solving techniques.
In [BB09], reactive search is characterized by the integration of machine learning techniques into search heuristics. A classification of the source of information that is used by the algorithm is proposed to distinguish between problem dependent information, task dependent information, and local properties.
In their survey [BHK+10], Burke et al. propose a classification of hyper-heuristics that are defined as 'search methods or learning mechanisms for selecting or generating heuristics to solve computational search problems'. As mentioned above, this classification also distinguishes between two dimensions: the different sources of feedback information and the nature of the heuristics search space. This classification is summarized in Figure 6.3.

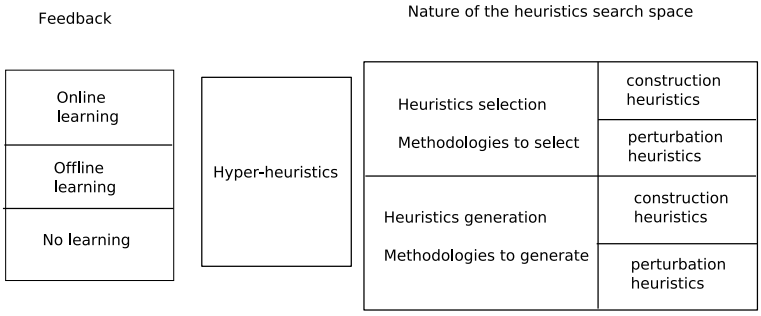


Figure 6.3: Classification of hyper-heuristics proposed by Burke et al. [BHK+10]

The feedback, when used, corresponds here to the information that is learned during solving (online) or using a set of training instances (offline). The authors identify two families of low level heuristics : construction heuristics (used to incrementally build a solution) and perturbation heuristics (used to iteratively improve a starting solution). The hyper-heuristics level can use heuristics selection methodologies, that produce combinations of pre-existing low level heuristics, or heuristics generation methodologies, that generate new heuristics from basic blocks of low level heuristics.
Another interesting classification is proposed in [GS08], in which Gagliolo et al. are interested in the algorithm selection problem [Ric75] and describe the different selection techniques according to the following points of views. The problem consists in assigning algorithms from a set of possible alternatives solving methods to a set of problem instances in order to improve the performance. Different

dimensions are identified with regards to this algorithm selection problem :

- The nature of the problems to be solved : decision vs. optimization problems.
- The generality of the selection process : selection of an algorithm for a set of instances or selection of an algorithm for each instance.
- The reactivity of the selection process : the selection can be static and made before running all the selected algorithms or can be dynamically adapted during execution.
- The feedback used by the selection process : the selection can be made from scratch or using previously acquired knowledge.
- The source of feedback: as in the previous classification, when learning is used in the selection process, one may consider offline (using separated training instances) or online (updating information during solving) learning techniques.

As claimed in the introduction, autonomous search aims at providing a more uniform description and characterization of these different trends, which have close relationships.

6.3 Architecture of Autonomous Solvers

We may define autonomous solvers as solvers that contain control in their search process (i.e., the solvers described in Section 6.4.2). We want to study such autonomous systems w.r.t. their specific control methods.
A general control process includes a strategy that manages the modification of some of the solver's components and behavioral features after the application of some solving functions. The overall strategy to combine and use components and parameters can be based on learning that uses information from the current solving process or from previous solved instances (see remarks in Section 6.2.6). Therefore, modifications are often based on a subset of search states. Given a solver, we have to consider the interactions between the heuristics and the strategy which selects the heuristics at a meta-level (notion of hyper-heuristics). On the one hand, one can consider the solver and its history and current environment (i.e., the previously computed search states and eventually other external information related to previous computations) as an experimental system, which is observed from an external point of view. Such a supervised approach then consists in correctly controlling the solver by adjusting its components according to criteria and decision rules (these rules may be automatically generated by means of statistics and machine learning tools or even by human experts). On the other hand, one may consider that the solver changes the environment at each step of the solving process and that this environment returns feedback information to the solver in order to manage its adaptation to this changing

context (different types of feedback may be taken into account as mentioned in Section 6.2.6). In this case, we will use self adaptation. To illustrate these ideas, we propose a high level picture of an autonomous search system (see Figure 6.4).

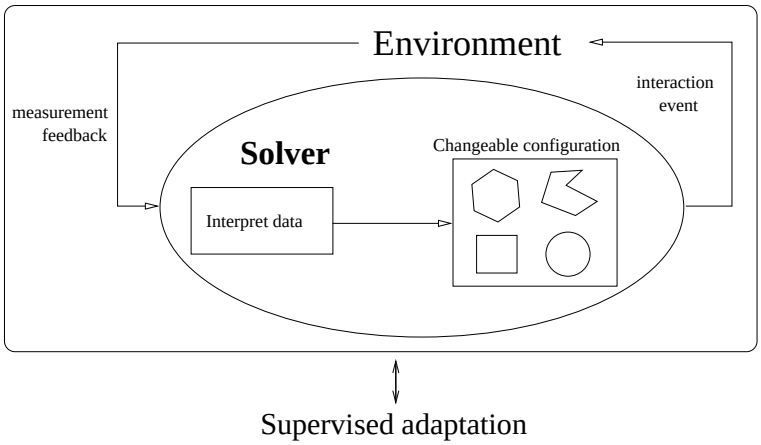


Figure 6.4: The global architecture of an Autonomous Search System

6.3.1 Control by Self Adaptation

In self adaptation, the adaptive mechanism is coupled with the search components, directly changing them in response to the consequences of their actions. Self-adaptive techniques are tightly integrated with the search process and should usually require little overhead. The algorithm is observing its own behavior in an online fashion, modifying its parameters accordingly. This information can be either directly collected on the problem or indirectly computed through the perceived efficiency of individual components. Because the adaptation is done online, there is an important trade-off between the time spent computing heuristic information and the gains that are to be expected from this information.

6.3.2 Control by Supervised Adaptation

Supervised adaptation works at a higher level. It is usually external and its mechanisms are not coupled with the search process. It can be seen as a monitor that observes the search and analyzes it. It can modify the components of the solver (or requires the solver to modify its components) in order to adapt it. Supervised adaptation can use more information, e.g., learning-based knowledge, etc. In some cases, we can imagine that typical supervised actions could be 'compiled' into self-adaptive mechanisms.

6.3.3 Searching for a Solution vs. Solutions for Searching

It appears now that the problem of building a good Autonomous Search solver is more ambitious than finding a solution to a given instance of a problem. Indeed, inspired by the seminal consideration of John Rice [Ric75] when he was abstracting the problem of finding the best algorithm for solving a given problem, we need to take into account at least three important spaces in which an autonomous search process takes place.

- The search space: the search space is induced by the encoding of the problem and corresponds to the set of all potential configurations of the problem that one has to consider in order to find a solution (or to find all solutions, or to find an optimal solution, ...). This search space can also be partitioned, for optimization problems, into the set of feasible solutions and infeasible solutions with respect to the constraints of the problem.

- The search landscape: the search landscape is related to the evaluation function that assigns a quality value to the elements of the search space. If indeed this notion is rather of limited use in the area of complete solvers, this is a crucial notion when using heuristics or metaheuristics, search algorithms whose purpose is to explore and exploit this landscape in order to find solutions. Most of the metaheuristics are designed accordingly to the management of this exploration-exploitation balance and the characteristics of the search landscapes, often use geographical metaphors: How to travel across plateaus? How to escape from a local optimum by climbing hills?, ...

- The algorithms' space: according to the previous description of solvers architecture, we have highlighted that a solver consists of components that define its structural properties together with a set of behavioral features (parameters and control rules). As mentioned before, given a basic algorithmic skeleton we may consider a set of possible solvers that correspond to the possible components choices and configurations. This algorithms' space can also be composed of different solvers when dealing with portfolio-based algorithm selection.

The relationships between these spaces are illustrated in Figure 6.5. Indeed, the ultimate autonomous search purpose can be formulated as: finding a suitable algorithm that is able to efficiently explore and exploit the search landscape in order to suitably manage the search space and find solutions to the initial problem.

6.3.4 A Rule-Based Characterization of Solvers

As already mentioned, the solving techniques used for solving such problems may include very different features from complete tree based solvers to local search or evolutionary algorithms. In this presentation, we will attempt to abstract

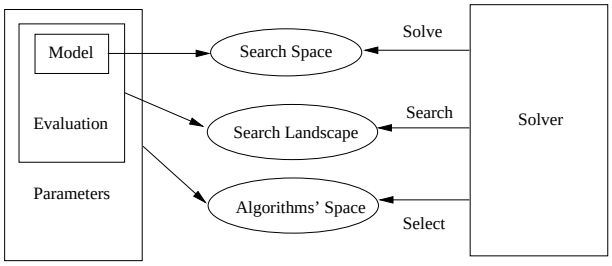


Figure 6.5: The solver and its action with respect to different spaces

theses solving features in order to be able to address general solving algorithms, focusing on their autonomous aspects as described above. Indeed, such rule-based formalizations have already been proposed for modeling some constraint programming solving processes [Apt03, FA03] and also for hybrid solvers including local search [MSL04]. Here, our purpose is not really to prove some properties of the solvers but rather to highlight their basic operational mechanisms in order to classify them with regards to their behavioral and structural characteristics.
When using a solver, one may distinguish two main tasks that correspond indeed to different but closely related levels of technical accuracy that can be achieved by more or less specialized users:

- The component design: this phase, consists in choosing the suitable components described in Section 6.2.3 that should be included in the solver with regards to the problem characteristics for instance. As mentioned above, these components constitute the architecture of the solver.
- The configuration of the solver through parameters settings and control: this second phase consists in defining through control features how the components can be used during the solving process.

Based on this consideration and on the general solver architecture depicted in Figure 6.1, we propose a formal description in the next section.

Formal Description

We define here some basic notions in order to characterize the behavior of solvers with a computationally oriented taxonomy. This approach will allow us to characterize the solvers. We first recall some basic concepts related to constraint satisfaction and optimization problems.

Definition 4. CSP
A CSP is a triple $(X, D, \mathcal{C})$*, where* $X = \{x_1, \cdots, x_n\}$ *is a set of variables whose values are restricted to given domains* $D = \{D_1, \cdots, D_n\}$*. There exists a bijective mapping that assigns each variable* x_i *to its corresponding domain, that*

will be noted D_{x_i}. We consider a set of constraints $\mathcal{C}$ as a set of relations over the variables X.

Definition 5. Search Space
The search space $\mathcal{S}$ is a subset of the possible configurations of the problem and can be the Cartesian product of domains $\Pi_{x \in X} D_x$. The choice of the internal representation (i.e., the model) defines the search space. An element s of the search space will be called a ***candidate solution****.*

Definition 6. Solution
A feasible solution is an assignment of values to variables, which can be seen as an element of $\mathcal{S}$ (i.e., given an assignment $\theta : X \rightarrow \Pi_{i=1}^{n} D_i$, $\theta(x_i) \in D_{x_i}$), and which satisfies all the constraints of $\mathcal{C}$. In the context of optimization problems, we also consider an objective function $f : \mathcal{S} \rightarrow \mathbb{R}$. An optimal solution is a feasible solution maximizing or minimizing, as appropriate, the function f.

We have now to define, according to Section 6.2, the different elements that are included in the solver.

Definition 7. Evaluation Functions
We denote by E the set of evaluation functions $e : \mathcal{S} \rightarrow \mathbb{R}$.

Definition 8. Parameters
We denote by P the set of parameters and a parametrization π is a mapping that assigns a value to each parameter. We denote by Π the set of parameterizations.

Definition 9. Solving operators
We denote by Ω a set of solving operators (operators for short) that are functions $o : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$.

Definition 10. Solving strategy
We denote by H the set of solving strategies that are functions $h : 2^{\mathcal{S}} \times \Pi \times E \rightarrow \Omega$.

For sake of simplicity, in the following we will refer to solving strategies as strategies. Solving strategies and solving operators are the key-points of the solving algorithm (see Figure 6.1): a strategy manages some operators to compute the solutions. We obtain:

Solving algorithm = solving strategy + solving operators

We now formalize the solving processes as transitions using rules over computation states.

Definition 11. Computation State
Given a CSP $(X, D, \mathcal{C})$, a search space $\mathcal{S}$, a set of operators Ω, a set of evaluation functions E, a set of parameters P and a set of solving strategies H, a computation state is a tuple $< O, \mathcal{S}, e, \pi, h | S >$ where:

- *$O \subseteq \Omega$, where O is the set of operators currently used in the solver,*

- $S \subseteq \mathcal{S}$, *is the current subset of candidate solutions,*
- $e \in E$, *is an evaluation function,*
- $\pi \in \Pi$ *is the current parametrization,*
- $h \in H$ *is the current solving strategy.*

Remarks:

- It is important to note that Ω, E, and H are sets that may not be yet computable. For example, H represents the set of all possible strategies, either already existing or that will be discovered by the solver (as defined in Definition 14). Similarly, all the operators of Ω are not known since they can be designed later by the solver. However, O is known and all its operators as well.
- S corresponds to the internal basic search structure: the search state. For instance, if we consider a genetic algorithm the search state will be a population. In the case of a complete backtracking solver, it will consist in an incomplete assignment, ...
- O is the current set of operators available in the solver at a given stage and that are extracted from a set Ω of potential operators that could be used in this solver. Indeed, some solvers may use new solving operators that are produced online or offline according to a general specification or according to design rules. Note that an operator allows the solver to perform a transition from one search state to another. This is therefore the key concept of the solving process and we want to keep it as general as possible to handle various solving paradigms (as mentioned above).
- The evaluation function e must evaluate the candidate solutions. This evaluation is used by the strategy in order to drive the basic solving task and by the control in order to drive the solver behavior.
- The solving strategy h will be used to select the suitable operator to apply on the current candidate solutions with respect to the current parametrization π and the evaluation function e.

Note that, for the sake of simplicity, we restrict ourselves to solvers that have only one evaluation function and one search space at a time. This is typically the case but this framework could be easily generalized to capture more 'exotic' situations.

We denote by CS the set of computation states. Note that a computation state corresponds in fact to a search state together with the current configuration of the solver.

Definition 12. Computation Rules
A computation rule is a rule $\frac{\sigma}{\sigma'}$ *where* σ *and* σ' *are computation states from* CS.

Identification of Computation Rules

We identify here specific families of computation rules with respect to the way they modify the computation states.

- **Solving:** The fundamental solving task of a classic solver consists in computing a new state from the current one according to a solving strategy that chooses the suitable operator to apply with respect to the current candidate solutions, the parametrization, and the evaluation function. This corresponds to the following rule:

  [Solv] Solving

  $$\frac{< O, \mathcal{S}, e, \pi, h|S >}{< O, \mathcal{S}, e, \pi, h|S' >}$$

  where $S' = o(S)$ and $o = h(S, \pi, e) \in O$.

- **Parametrization:** The modification of the solver's parameters changes its configuration and can be used either to tune the solver before running it or to adjust its behavior during the run. A parametrization rule can be abstracted as:

  [Par] Parametrization

  $$\frac{< O, \mathcal{S}, e, \pi, h|S >}{< O, \mathcal{S}, e, \pi', h|S >}$$

- **Evaluation function modification:** Since we address here autonomous systems that are able to modify not only their configuration trough their parameters but also their internal components, we have to consider more intricate rules. A first way to adapt the solver to changes is to modify its evaluation function, which directly induces changes on the search landscape. This is the case when changing weights or penalties in the evaluation function (there are many examples, for instance [KP98, PH06]).

  [EvalCh] Evaluation modification

  $$\frac{< O, \mathcal{S}, e, \pi, h|S >}{< O, \mathcal{S}, e', \pi, h|S >}$$

- **Operators modification:** Another possibility to modify the internal configuration of the solver is to change its set of operators. Note that

operators can be added or discarded from the set O.

[OpCh] Operators modification

$$\frac{< O, \mathcal{S}, e, \pi, h|S >}{< O', \mathcal{S}, e, \pi, h|S >}$$

- **Strategy modification:** Similarly, solving strategies can be changed to manage differently the operators and achieve a different solving algorithm. As mentioned above, a backtracking algorithm can applied different strategy for enforcing local consistency at each node, or in hybrid solving one may switch from complete to approximate methods.

[StratCh] Strategy modification

$$\frac{< O, \mathcal{S}, e, \pi, h|S >}{< O, \mathcal{S}, e, \pi, h'|S >}$$

- **Encoding modification:** We also have to take into account solvers that will be able to change their encoding during execution. As this is the case for the evaluation modification, such changes will affect the search landscape.

[EncCh] Encoding modification

$$\frac{< O, \mathcal{S}, e, P, h|S >}{< O, \mathcal{S}', e, P, h|S >}$$

Note that applying one of these rules (except [Res]) will generally require applying other computation rules. For example, a change of encoding ([EncCh]) will certainly require a change of operators ([OpCh]), of evaluation function ([EvalCh]), of strategy ([StratCh]), and of parametrization ([Par]). However, a change of strategy does not always imply a change of operators.

Control of the Computation Rules and Solvers

The most important part of our characterization concerns the control of the algorithm to finally build the solver. The control is used to act on the configuration of the solver through its parameters but also to modify the internal components of the solver (parameters, operators, strategies, ...).

Definition 13. Control
Let $\mathcal{S}_{CS}$ be the set of all the finite sequences of elements of CS. A control function $K : \mathcal{S}_{CS} \rightarrow R$ is a function that selects a computation rule from the set R according to a sequence of computation states.

A solver state can be defined by a set of computation rules, and a sequence of computation states that have been previously computed.

Definition 14. Solver
A solver is a pair (K, R) composed of a control function K and a set of computation rules R that will define a sequence of solver states.

A way of describing a solver is to use regular expressions which schedule computation rules to describe its control. Let's come back to the rules defined in Section 6.3.4. We consider the set of rules $R = Par \cup Res \cup EvalCh \cup EncCh \cup OpCh \cup StratCh$ where Par represents some parametrization rules $[Par]$, $EvalCh$ some evaluation modification rules $[EvalCh]$, ... Given two subsets R_1 and R_2 of R, R_1^* means that zero or more rules of R_1 are sequentially applied and R_1R_2 means the sequential application of one rule of the subset R_1 followed by the application of one rule of R_2. $R_1|R_2$ corresponds to use of one rule from R_1 or one from R_2. These notations will be used in the following section to highlight the characteristics of the solvers by means of the sequences of rules that they apply in their solving processes.

Definition 15. Solver State
A solver state is a pair (R, Σ) where:

- *R is a set of computation rules as defined above*
- *Σ is a sequence of computation states that are recorded along the solving process.*

Starting from a solver state (R, Σ), with $\Sigma = (\sigma_0, \cdots, \sigma_n)$ the next state is obtained as (R, Σ') where $\exists r \in R$, such that $K(\Sigma) = r$ and $\Sigma' = (\sigma_0, \cdots, \sigma_n, \sigma_{n+1} = r(\sigma_n))$.

Note that in practice, a solver state does not contain the complete history. Thus, the sequence of computation states is either limited to a given length, or only the most relevant computation states are kept.
We now have:

Solver = Control + Configured Solving Algorithms

We recall that we stated before that Solving algorithm = Solving Strategy + Solving Operators. Coming back to Figure 6.3 that shows a classification of hyper-heuristics, we can notice that we obtain similar distinction here: solvers correspond to the hyper-heuristics of Figure 6.3, solving algorithms to heuristics search space, strategies to heuristics selection or generation, and operators to construction or perturbation heuristics. We can finally identify an autonomous solver:

Definition 16. Autonomous Solver
Consider a solver given by a regular expression ex of computation rules from $R = Par \cup Solv \cup EvalCh \cup EncCh \cup OpCh \cup StratCh$. *A solver is autonomous if ex contains at least a rule from* $Par \cup EvalCh \cup EncCh \cup OpCh \cup StratCh$ *(i.e., ex is not only composed of rules from Solv).*

An autonomous solver is a solver that modifies its configuration during solving, using a control rule. Of course, there are various degrees in this autonomy scale. We can now come back to the previous taxonomy of offline/tuning and online/-control (e.g., for parameters). Consider a solver given by a regular expression ex of computation rules from $R = Par \cup Solv \cup EvalCh \cup EncCh \cup OpCh \cup StratCh$, and the word w given by flattening this expression ex. The offline/tuning of a solver consists of the rules that appear in ex before the first Solv rule of ex. The online/control is composed of all the rules that appear after the first rule Solv and that are not of the Solv family of rules.
In the next section we will illustrate how these rules are used in real solvers and how they can be used to characterize families of solvers within our autonomous search scope.

6.4 Case Studies

In this section, we will not attempt to present an exhaustive view of existing solvers but we will rather choose some representative solvers or algorithms in order to illustrate different solving approaches and how the previous computation rules can be used to characterize these approaches. As mentioned in the introduction, autonomous search has been indeed investigated for many years, across many different areas and under different names. Therefore, we could not imagine providing an exhaustive discussion of all approaches.

6.4.1 Tuning Before Solving

As in [EHM99, LLM07], we use the word tuning for the adjustment of the different components of the algorithm before trying to solve an instance (see end of Section 6.3.4).

Preprocessing techniques

Even if preprocessing is not directly linked to the core of the solving mechanism but relies on external processes, we have to consider it as an important component in the design of modern solvers. Nowadays, efficient solvers (e.g. DPLL) use simplification preprocessing before trying to solve an instance (see for instance the SAT solver SatElite [EMS07]). Note that the model transformation can maintain equisatisfiability or a stronger equivalence property (the set of solutions is preserved).

Parameter Tuning on Preliminary Experiments

Such a tuning phase may consist in setting correct parameters in order to adjust the configuration of the solver. Here, these settings are performed according to a given set of preliminary experiments. Tuning before solving, will correspond to the configuration of the solver and then its use for properly solving the problem. Therefore, the general profile of the solvers will be mainly described as :

$$[Config]Solv^*$$

where $[Config]$ is of the form $(Par|EvalCh|OpCh|EncCh)^*$.

Empirical Manual Tuning: we include in this family the classic tuning task involved when using single metaheuristics based solvers where experiments are required to tune the various parameters [SE09, NSE08]. Of course there exist similar studies in constraint programming to choose the suitable variable and value choice heuristics, and this task is often not formalized. Most of the time, parameters are tuned independently since it appears difficult to control their mutual interaction without a sophisticated model. Here, the parametrization is not really part of the solver but rather a preliminary experimental process.

$$\textit{Solver: } Solv^*$$

Deciding the Size of a Tabu List: experiments or other previous analysis can be used to extract general parameters or heuristics' settings. In the context of Tabu Search for SAT, [MSG97] have used an extensive offline experimental step to determine the optimal length of a tabu list. They used simple regression to derive the length of the list according to the number of variables n. Remarkably, the length is independent of the size of the constraints, and their formula applies to any hard-random k-SAT instance. Therefore the parametrization can be included as a first step of the solving process.

$$\textit{Solver: } Par\_Solv^*$$

Automatic Parameter Tuning by an External Algorithm: recently, [HHS07] proposed an algorithm to search for the best parameters in the parameter space and therefore to automatically tune a solver. Now, if we consider that this automated process is included in the solver, we have then the following description.

$$\textit{Solver: } (Solv^*Par)^*Solv^*$$

Note that $(Solv^*Par)^*$ corresponds to a series of runs and parameter tuning, which is achieved automatically.

Components Setting Before Solving

We consider here methods that consist in choosing the correct components of the solver by using experiments and/or external knowledge that has been acquired apart from the current solving task. This knowledge can be formulated as

general rules, can use more or less sophisticated learning techniques, or may also use an external computation process.

A. Learning Solver's Components External mechanisms can be used before tuning to discover or learn efficient components for the solver.

Discovering Heuristics: in [Fuk08], genetic programming is used to discover new efficient variable selection heuristics for SAT solving with local search algorithms. Candidate variable selection heuristics are evaluated on a set of test instances. This automatic process can be inserted before solving (the variable selection heuristics can induce a change of parameters or operators depending on the description granularity). Note that here the first $Solv^*$ is not applied to the problem at hand.

$$\textit{Solver: } (Solv^*(OpCh|Par))^*Solv^*$$

The choice heuristics can be parameters of the operators in our formalism, heuristics discovering can be considered as the selection of suitable operators and their parametrization.

Learning Evaluation Functions: in [BMK00], a new method is proposed in order to learn evaluation functions in local search algorithms and improve search efficiency based on previous runs.

$$\textit{Solver: } (Solv^*EvalCh)^*Solv^*$$

B. Empirical Prediction of Instances Hardness
The following techniques are based on a learning component (e.g., clustering tools), which can be used to detect automatically the suitable heuristics and strategies to apply.

Portfolio-based: in SATzilla [XHHLB08], offline linear basis function regression and classifiers are used on top of instances-based features to obtain models of SAT solvers runtime. During the exploitation phase, instances features are used to select the best algorithm from a portfolio of tree and local search based SAT solvers. We may also cite the works of Gebruers et al. [GGHM04] and Guerri et al. [GM04] that use case based reasoning and learning techniques to choose the appropriate solving technique among constraint programming and integer linear programming. In these solvers schemes, the first $Solv^*$ corresponds again to preliminary experiments.

$$\textit{Solver: } Solv^*(OpCh|StratCh|Par|EvalCh)^*Solv^*$$

Parameter-based: in [HH05, HHHLB06], the authors use an approach similar to SATzilla. They showed that it is possible to predict the runtime of two stochastic local searches (SLS). In this work, the selection of the best method to apply on a given instance is changed into the selection of the best parameters of a given SLS algorithm.

$$\textit{Solver: } ParSolv^*$$

6.4.2 Control During Solving

The control of the solver's behavior during the run can be achieved by either modifying its components and/or its parameters. This corresponds, for instance, to an online adjustment of the parameters or heuristics. Such control can be achieved by means of supervised control schemes or by self adaptive rules. Of course, such approaches often rely on a learning process that tries to benefit from previously encountered problems along the search or even during the solving of other problems. Therefore, the profile of the solvers will generally be:

$$([Config]Solv^*)^*$$

where $[Config]$ is of the form $(Par|EvalCh|OpCh|EncCh)^*$. Note that the outter * loop represents indeed the control loop.

Controlling Encoding

[Han08] proposes an adaptive encoding in an evolutionary algorithm in order to solve continuous function optimization problems. The representation of the solutions are changed along the search to reach an optimal representation that could simplify the solving of the initial problem.

$$\textit{Solver: } (EncChSolv^*)^*$$

Controlling Variable Orderings and Values Selection in Search Heuristics

We consider here approaches where the heuristics functions change during the search w.r.t. the current state and parameters.

Hybrid Approaches to Discover Efficient Variable Ordering: To illustrate this kind of approach, we may mention the SAT solving technique of [MSG98] where a Tabu Search is used at each node of a DPLL to find the next variable to branch on.

$$\textit{Solver: } ((OpChStratCh)Solv^*ParSolv^*)^*$$

Continuous Search: in [AHS10], the authors propose to exploit the result of an offline learning stage to select the best variable and value heuristics. They use a restart-based tree-search algorithm and tune the previous heuristics at each new restart point. Moreover, this approach perpetually refines its learning stage by reassessing its past choices in between successive calls to the search procedure. This approach is presented in Chapter 5.

$$\textit{Solver: } (ParSolv)^*$$

A Conflict-driven Heuristic: in [BHLS04a], important variables are deemed to be the ones linked to constraints that have frequently participated in dead-ends. During the search, this information is collected and used to order variables. Eventually, the system has enough knowledge to branch on important variables

and quickly solve the problem. The system 'learns' weights from conflicts that are used in the computation of the variable selection heuristics, this corresponds to an update of the parameters each time a conflict is met.

$$Solver: (ParSolv^*)^*$$

A variable-dependency based Heuristic: in [AH09], the constraint propagation engine is exploited to detect so called weak-dependencies between variables. These correspond to situations when the instantiation of a given variable leads to the instantiation of others. These events are perceived as positive, and are used to rank the variables, favoring the ones whose branching-on results in the largest number of instantiation. This heuristic is shown to outperform [BHLS04a] on many domains.

$$Solver: (ParSolv^*)^*$$

Implicit feed-back loops in modern DPLL solvers: in modern SAT solvers like the one presented in [ES03a], many implicit feed-back loops are used. For instance, the collect of conflicts feeds the variable selection heuristic, and the quality of unit propagation is sometimes used to control the restart strategy. Similarly, the deletion of learned clauses which is necessary to preserve performances uses activity-based heuristics that can point to the clauses that were the least useful for the unit propagation engine. Therefore, it induces changes in the model itself and in the heuristics parameters.

$$Solver: ((EncCh|Par)Solv^*)^*$$

Adapting Neighborhood During the Search: variable neighborhood search [MH97, HR06, PR08] consists in managing simultaneously several neighborhood functions and or parameters (according to the description granularity) in order to benefit from various exploration/exploitation facilities.

$$Solver: ((OpCh|Par)Solv)^*$$

Evolving Heuristics

Hyper-heuristics: hyper-heuristics [BGJ+03] is a general approach that consists in managing several meta heuristics search methods from a higher strategy point of view. Therefore, it is closely related to autonomous search and has already been applied for many problems (e.g., SAT solving [BEDP08]). Since they switch from one solving technique to another, hyper-heuristics could be characterized by :

$$Solver: ((OpCh|StratCh|Par|EvalCh)^*Solv^*)^*$$

Learning Combinations of Well-known Heuristics: in the ACE project [EFW05], learning is used to define new domain-based weighted combinations of branching heuristics (for variable and value selection). ACE learns the weights to apply through a voting mechanism. Each low-level heuristic votes for a particular element of the problem (variable, value). Weights are updated according to

the nature of the run (successful or not). The learning is applied to a given class of problems. The combination is learned on a set of representative instances and used during the exploitation step. A similar approach has been used in [GJ08] in order to learn efficient reduction operators when solving numerical CSPs.

$$\textit{Solver: } (ParSolv^*)^*$$

Controlling Evaluation Function

This aspect may concern local search algorithms that use for instance adaptive weighting of the constraints in their evaluation function [Mor93, Tho00]. Constraint weighting schemes solve the problem of local minima by adding weights to the cost of violated constraints. These weights increase the cost of violating a constraint and so change the shape of the cost surface w.r.t. the evaluation function. Note that these techniques are also widely used in SAT solvers [BHvMW09].

$$\textit{Solver: } (EvalChSolv^*)^*$$

Parameters Control in Metaheuristics Algorithms

We consider here approaches that change the parameters during the search w.r.t. the current state and other parameters. Of course, these parameters have a direct influence on the heuristics functions, but these latter functions stay the same during the solving process.

Reactive Search: in [BBM08] (formerly presented in [BBM07]), Battiti et al. propose a survey of so-called reactive search techniques, highlighting the relationship between machine learning and optimization processes. In reactive search, feedback mechanisms are able to modify the search parameters according to the efficiency of the search process. For instance, the balance between intensification and diversification can be automated by exploiting the recent past of the search process through dedicated learning techniques.

$$\textit{Solver: } (ParSolv^*)^*$$

Adaptive Genetic Algorithms: adaptability is well-known in evolutionary algorithms design. For instance, there are classical strategies to dynamically compute the usage probability of GA search operators [Thi05, WPS06a, WLLH03]. Given a set of search operators, an adaptive method has the task of setting the usage probability of each operator. When an operator is used, a reward is returned. Since the environment is non-stationary during evolution, an estimate of the expected reward for each operator is only reliable over a short period of time [WPS06b]. This is addressed by introducing a quality function, defined such that past rewards influence operator quality by an extent that decays exponentially with time. We may also mention other works that use more

sophisticated evaluation functions, rewards computation and operator probability adjustment in order to manage dynamically the application parameters of the EA [MFS+09, MS08, FDSS08].

$$\textit{Solver: } (ParSolv^*)^*$$

6.4.3 Control During Solving in Parallel and Distributed Search

The solvers described in this section also belong to the previous family of solvers that include control within their proper solving process. But here, due to the parallel/distributed architecture of solver, the sequence of computation rules is more difficult to schedule. Thus, the profile could be described as $([Config]|Solv^*)^*$

Value-ordering in Portfolio-based Distributed Search: in [RH05], the authors present a portfolio-based distributed search. The system allows the parallel execution of several agent-based distributed search. Each search requires the cooperation of a set of agents which coordinate their local decisions through message passing. An agent is part of multiple distributed search, and maintains the context of each one. Each agent can aggregate its context to dynamically rank the values of its local variables. The authors define several efficient portfolio-based value-ordering heuristics. For instance, one agent can pick up the value which is used most frequently in competing search, or the one which is most supported in other searches, etc. This approach is fully described in Chapter 3.

$$\textit{Solver: } (Par|Solv^*)^*$$

Adaptive Load-balancing Policies in Parallel Tree-based Search: Disolver is an advanced Constraint Programming library which particularly targets parallel search [Ham03]. This search engine is able to dynamically adapt its inter-processes knowledge-sharing activities (load balancing, bound sharing). In Disolver, the end-user can define constraint-based knowledge sharing policies by adding new constraints. This second modeling can be linked to the constraint-based formulation of the problem to control the knowledge sharing according to the evolution of some problem components. For instance, the current value of the objective function can be used to allow answers to incoming load-balancing requests when the quality of the current subtree is perceived as good, etc. Interestingly, since the control of the knowledge sharing policies is made through classical constraints, it is automatically performed by the constraint propagation engine. We can see this as a dynamic adjustment of knowledge sharing activities, and assimilate it to model (learned clauses) and parameters (selection heuristics) change.

$$\textit{Solver: } ((EncCh|Par)|Solv^*)^*$$

Control-based Clause Sharing in Parallel SAT Solving: Conflict driven clause learning, one of the most important component of modern DPLL, is crucial to the performance of parallel SAT solvers. Indeed, this mechanism allows clause sharing between multiple processing units working on related (sub-)problems. However, without limitation, sharing clauses might lead to an exponential blow up in communication or to the sharing of irrelevant clauses. In [HJS09], the authors propose new innovative policies to dynamically select shared clauses in parallel solvers. The first policy controls the overall number of exchanged clauses whereas the second one additionally exploits the relevance or quality of the clauses. This dynamic adaptation mechanism allows to reinforce/reduce the cooperation between different solvers which are working on the same SAT instance. This approach is fully described in Chapter 4.

$$Solver{:}(Par|Solv^*)^*$$

6.5 Summary

In this chapter, we have proposed a taxonomy of search processes w.r.t. their computation characteristics. To this end, we have presented the general basic concepts of a solver architecture: the basic components of a solver, and its configurations. We have then identified autonomous solvers as solvers that can control their solving process, either by self adaptation (internal process) or by supervised adaptation (external process).
We have proposed a rule-based characterization of autonomous solvers: the idea is to formalize solvers adaptations and modifications with some computation rules that describe solver transformation. Using our formalism, we could then classify, characterize, and identify in the scope of autonomous search representative solvers by outlining their global mechanism.

Our description framework allows us to handle solving techniques:

- of various and different types: either complete, incomplete, or hybrid,
- based on different computation paradigms: sequential, distributed, or parallel
- dedicated to different problem families: CSP, SAT, optimization, ...

This work was also an attempt to highlight the links and similarities between different communities that aim at building such autonomous solvers and that may benefit from more exchanges and more collaborative approaches (including constraint programming, SAT, machine learning, numerical optimization, clustering, ...).

We have identified the notion of control in autonomous constraint solvers and two main techniques for achieving it: control by supervised adaptation and

control by self-adaptation, depending on the level of interaction between the solver, its environment, and the control itself. These two control management approaches are indeed complementary. Moreover, they open new challenges for the design of more autonomous search systems that would run continuously, alternating (or combining, or executing in parallel) solving and self-improving phases. A first attempt in this direction has been presented in Chapter 5.

Chapter 7

Conclusion and Perspectives

The writing of this document gave me the occasion to put my work into perspectives and to reassess its homogeneity and consistency. Clearly, my work on distributed constraint satisfaction put me on the distributed system side very early. In that world, algorithms are more than monolithic sets of instructions and have value in their well timed and controlled interactions.

I decided to exploit the richness of this setting to mitigate the risk of being wrong in a constructive search process. Initially by adding parallelism to distributed search [Ham99b, Ham02b], then as presented in Chapter 3 by organizing competition and cooperation between multiple distributed search strategies. Competition is rather straight forward to organize. On the other hand, cooperation opens a new space where the benefit of the knowledge exchanged has to be balanced against the cost of sharing knowledge. When information is shared, we have to consider the ramp-up time to prepare information, and the time it takes to effectively exchange the information. When information is not shared we have to consider that costly redundant work can occur, and in divide-and-conquer systems, that task starvation can happen.

Therefore, controlling the way knowledge is shared and which knowledge is exchanged is crucial to the performance. In DisCSP settings, we managed to exploit agents' locality to share information between strategies. This allowed exchange at virtually no cost. Concerning the knowledge to share, we tried to be systematic by exploring policies based on diversification and emulation principles.

In the parallel SAT settings, modern solvers allow the exchange of conflict-clauses. However, since they can generate millions of clauses during their effort, the exchange has to be well controlled. Technically, we decided to exploit lockless data structures to maximize performance. Content wise, we managed to develop new techniques to assess the quality of conflict-clauses in an attempt to exchange meaningful information. We got inspired by control-theory techniques to finely tune the exchanges.

When one cannot multiply search strategies to avoid being wrong, the selection of the right strategy is crucial. One way to avoid mistakes is to offline tune a predictive model which accurately matches instances features to good solver's parameters [HH05, HHHLB06]. This approach requires a good understanding of the application domain and a large set of representative instances. This last requirement can be dropped by streamlining the learning process between executions of the search procedure. Since the learning is performed on "real" instances, the model is more accurate. As a downturn, such a system cannot give top performance with the first instances but can only gradually improve over time. Such a Continuous Search system was presented in Chapter 5.

Finally, to capture our contributions in an unifying framework which will also embed related work as much of possible, we moved to the notion of Autonomous

Search. We defined autonomous solvers as solvers that contain control in their search process, and study such autonomous systems w.r.t. their specific control methods. A control process includes a strategy that manages the modification of some of the solver's components and behavioral features after the application of some solving functions. We gave a formalizing of solvers adaptations and modifications with computation rules that describe the modification of the solver's components transformation.

In the following, we discuss some of the main challenges for each sub domains.

Distributed Constraint Networks

Our present results greatly improve the applicability of DisCSP algorithms by providing greater efficiency and robustness to two classical tree search algorithms. In future work we would like to investigate how portfolios are best composed and how they could implement a more informed Aggregation (beyond agent's scope). The composition could be studied with different hand or system made portfolios or by dynamic adaptation during search. The latter could provide more resources to the most promising efforts. The former could take advantage of heterogeneous portfolios involving various tree and local-search combined with some distributed consistency-enforcement method (e.g., [Ham99a, Ham02c]). Finally, knowledge Aggregation could be easily improved at no cost by adding extra information to existing message passing operations (search effort, etc). This would give a better view of the distributed system, and could benefit from new aggregation methods.

Parallel SAT Solving

Clearly, the parallel portfolio approach for SAT has been adopted by the whole SAT community [Bie10, Kot10, SLB10]. This approach came as a surprise at the 2008 SAT-Race. Until then, the divide-and-conquer approach seemed the only way for many people. Parallel portfolios exploit the fact that modern SAT solvers are highly stochastic and are worth differentiating for better performances. More crucially, they benefit from a crucial property of modern SAT solvers: they do not need to exhaust a search space to definitely qualify an input as satisfiable or not. We came up with the ManySAT architecture thanks to our early experience on distributed portfolios, and thanks to our experience of parallel divide-and-conquer in constraint programming [Ham03].
As future work, the question of the scalability of parallel SAT portfolios able to exchange conflict-clauses has to be asked. Many attempts have been done to mix portfolios and divide-and-conquer approaches [MML10], however the results so far are not convincing.
Here is one big challenge for parallel search in general: being able to scale to very large search systems while maintaining the ability to exchange knowledge opportunistically and efficiently[1]. I am confident that Parallel SAT solvers will

[1]See [HW12] for other challenges in Parallel SAT Solving.

improve in the near future, notably, through regular competitions [JBRS12].

Continuous Search

Continuous computation addresses the issue not of finding the best (boundedly optimal) use of time in solving a given problem, but the best use of idle computational resources between bouts of problem solving. This approach broadens the definition of a 'problem' to include not just individual instances, but the class of challenges that a given computational system is expected to face its lifetime. Eventually, the end of the current search is just another event for the AS system. As an effect, the priority of its long lasting self-improving task is raised and the task becomes foreground. The latest resolution is here to enrich the knowledge of the system and is eventually exploited during this new task. We can envision a wide range of actions that can be overtaken by the search algorithm while it is idle:

- Analyzing the strategies that have succeeded and failed during the last runs.
- Performing costly machine learning techniques in order to improve a supervised tuning method.
- Using knowledge compilation techniques in order to compile new deduction rules, or new patterns that were detected in the recently solved problems and that can prove useful for future problems of the same application area.
- Exchange gained knowledge with similar AS systems, e.g., features-based prediction function.

In fact, such a continuous system would include a self-adaptive strategy during the solving process while it could switch to a supervised controller while waiting for another problem instance. This architecture would allow it to react dynamically to incoming events during solving and to exploit the knowledge acquired through its successive experiences.
The performance evaluation of an AS able to work in continuous search mode is also an important problem which is highly related to the arrival rate and to the quality of new problem instances. Here quality corresponds on how good the instances are for the AS to gain important knowledge on the whole problem class.

Autonomous Search

An important issue is evaluating performances of Autonomous Search systems with respect to or compared to classical criteria, used in solver competitions for instance. We think that the performance evaluation of an autonomous search may actually focus on three points:

- show that an autonomous search can (re)discover the best known or approximate a very good strategy for a specific problem,
- show the ability of an autonomous search to adapt itself to a changing environment, e.g., more or less computational resources,
- show that an autonomous search could adapt itself and converge to an efficient strategy for a class of problems.

There exists an optimal search strategy for a particular problem. However, determining such strategy could require much more computational power than solving the problem at hand. One possible way to assess the performance of AS systems is to run them on artificial problems where the optimal strategy is well known and to see if their adaptive mechanisms are able to build a strategy close to the optimal.
The efficiency of an AS system can also be measured as its ability to maintain the competitiveness of its search strategy in a changing environment. Here, the goal is more to assess the reaction-time of the system under changing settings rather than the ultimate quality of the produced strategies.
A major challenge associated to AS is that classical tools for algorithm analysis typically provide weak support for understanding the performance of autonomous algorithms. This is because autonomous algorithms exhibit a complex behavior that is not often amenable to a worst/average -case analysis. Instead, autonomous algorithms should be considered as full-fledged complex systems, and studied as such.

Bibliography

[ABCC07] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, January 2007.

[ABH+08] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A generalized framework for conflict analysis. In Büning and Zhao [BZ08], pages 21–27.

[AD97] A. Armstrong and E. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proc. of the 15th Int. Joint Conf. on AI (IJCAI-97)*, pages 620–625, 1997.

[AH09] Alejandro Arbelaez and Youssef Hamadi. Exploiting weak dependencies in tree-based search. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 1385–1391. ACM, 2009.

[AHS09] Alejandro Arbelaez, Youssef Hamadi, and Michele Sebag. Online heuristic selection in constraint programming. In *International Symposium on Combinatorial Search (SoCS)*, Lake Arrowhead, USA, July 2009.

[AHS10] Alejandro Arbelaez, Youssef Hamadi, and Michèle Sebag. Continuous search in constraint programming. In *ICTAI (1)* [DBL10], pages 53–60.

[AKJ04] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. In *ECML*, volume 3201 of *LNCS*, pages 39–50, Pisa, Italy, Sept 2004. Springer.

[AL03] E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[Apt03] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[AST09] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *CP*, volume 5732 of *LNCS*, pages 142–157, Lisbon, Portugal, Sept 2009. Springer.

[BB09] R. Battiti and M. Brunato. *Handbook of Metaheuristics (2nd edition)*, chapter Reactive Search Optimization: Learning while Optimizing. Springer, 2009. In press.

[BBM07] R. Battiti, M. Brunato, and F. Mascia. Reactive search and intelligent optimization. Technical report, Dipartimento di Informatica e Telecomunicazioni, Univerita di Tranto, Italy, 2007.

[BBM08] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations research/Computer Science Interfaces*. Springer Verlag, 2008.

[BBMM05] C. Bessiere, I. Brito, A. Maestre, and P. Meseguer. Asynchronous backtracking without adding links: A new member in the ABT family. *Artificial Intelligence*, 161:7–24, 2005.

[BEDP08] M. Bader-El-Den and R. Poli. Generating SAT local-search heuristics using a gp hyper-heuristic framework, artificial evolution. In *8th International Conference, Evolution Artificielle, EA 2007. Revised Selected Papers*, number 4926 in Lecture Notes in Computer Science, pages 37–49. Springer, 2008.

[BFH+05] Mark Bartlett, Alan M. Frisch, Youssef Hamadi, Ian Miguel, Armagan Tarim, and Chris Unsworth. The temporal knapsack problem and its solution. In Roman Barták and Michela Milano, editors, *CPAIOR*, volume 3524 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2005.

[BG06] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 16. Elsevier, 2006.

[BGJ+03] E.K. Burke, G.Kendall, J.Newall, E.Hart, P.Ross, and S.Schulenburg. *Handbook of Meta-heuristics*, chapter Hyper-heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer, 2003.

[BGS99] L. Brisoux, E. Grégoire, and L. Sais. Improving backtrack search for SAT by means of redundancy. In *Foundations of Intelligent Systems, 11th International Symposium, ISMIS '99*, volume 1609 of *Lecture Notes in Computer Science*, pages 301–309. Springer, 1999.

[BHK+09] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. A survey of hyper-heuristics. Technical Report Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham, Computer Science, 2009.

[BHK+10] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. *Handbook of Meta-heuristics2nd Edition*, chapter A Classification of Hyper-heuristics Approaches, pages 449–468. Springer, 2010.

[BHLS04a] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In R. López de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 146–150. IOS Press, 2004.

[BHLS04b] Frederic Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150, Valencia, Spain, Aug 2004. IOS Press.

[BHS09] Lucas Bordeaux, Youssef Hamadi, and Horst Samulowitz. Experiments with massively parallel constraint solving. In Boutilier [Bou09], pages 443–448.

[BHvMW09] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[BHZ06] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Survey*, 9(2):135–196, 2006.

[Bie08] A. Biere. Adaptive restart strategies for conflict driven SAT solvers. In Büning and Zhao [BZ08], pages 28–33.

[Bie10] A. Biere. Lingeling, plingeling, picosat and precosat at SAT race 2010. Technical Report 10/1, FMV Reports Series, 2010.

[BMK00] J. Boyan, A. Moore, and P. Kaelbling. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:200–0, 2000.

[Bou09] Craig Boutilier, editor. *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 2009.

[BR96] C. Bessiere and J. C. Regin. Mac and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *CP*, pages 61–75, 1996.

[BS96] M. Böhm and E. Speckenmeyer. A fast parallel SAT-solver - efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17(3-4):381–400, 1996.

[BSK03] W. Blochinger, C. Sinz, and W. Küchlin. Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing*, 29(7):969–994, 2003.

[BSPV02] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann Publishers, 2002.

[BT94] R. Battiti and G. Tecchiolli. The reactive tabu search. *INFORMS Journal on Computing*, 6(2):126–140, 1994.

[Buc06] B. G. Buchanan. What do we know about knowledge? *AI magazine*, 27(4):35–46, 2006.

[BZ08] H. K. Büning and X. Zhao, editors. *Theory and Applications of Satisfiability Testing - 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*. Springer, 2008.

[CB01] J. Crispim and J. Brandão. Reactive tabu search and variable neighbourhood descent applied to the vehicle routing problem with backhauls. In *Proceedings of the 4th Metaheuristics International Conference, Porto, MIC 2001*, pages 631–636, 2001.

[CB04] Tom Carchrae and J. Christopher Beck. Low-knowledge algorithm control. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 49–54. AAAI Press / The MIT Press, 2004.

[CB05] Tom Carchrae and J. Christopher Beck. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence*, 21(4):372–387, 2005.

[CB08] Marco Correira and Pedro Barahona. On the efficiency of impact based heuristics. In *CP*, volume 5202 of *LNCS*, pages 608–612, Sydney, Australia, Sept 2008. Springer.

[CBH+07] F. Corblin, L. Bordeaux, Y. Hamadi, E. Fanchon, and L. Trilling. A SAT-based approach to decipher gene regulatory networks. In *Integrative Post-Genomics, RIAMS, Lyon*, 2007.

[CGTT63] W. Crowston, F. Glover, G. Thompson, and J. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. Technical report, ONR Research Memorandum No. 117, GSIA, Carnegie-Mellon University, Pittsburg, PA, 1963.

[CH06] Yek Loong Chong and Youssef Hamadi. Distributed log-based reconciliation. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 108–112. IOS Press, 2006.

[CKS02] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Applications of Evolutionary Computing, EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *TOCS*, 3(1):63–75, Feb 1985.

[CS00] P. Cowling and E. Soubeiga. Neighborhood structures for personnel scheduling: A summit meeting scheduling problem (abstract). In E. K. Burke and W. Erben W., editors, *proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Constance, Germany, 2000.

[CS08] G. Chu and P. J. Stuckey. Pminisat: a parallelization of minisat 2.0. Technical report, Sat-race 2008, solver description, 2008.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[CW03] W. Chrabakh and R. Wolski. GrADSAT: A parallel SAT solver for the grid. Technical report, UCSB Computer Science Technical Report Number 2003-05, 2003.

[DBL09] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2009, Trondheim, Norway, 18-21 May, 2009*. IEEE, 2009.

[DBL10] *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 1*. IEEE Computer Society, 2010.

[DD01] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'01*, pages 248–253, 2001.

[De 06] K. De Jong. *Evolutionary computation: a unified approach*. MIT Press, 2006.

[Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[DLL62] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[dMB08] L. Mendonça de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[EB05] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *Theory and Applications of Satisfiability Testing, SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[EFW+02] S. Epstein, E. Freuder, R. Wallace, A. Morozov, and B. Samuels. The adaptive constraint engine. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference*, volume 2470 of *Lecture Notes in Computer Science*, pages 525–542. Springer, 2002.

[EFW05] S. Epstein, E. Freuder, and R. Wallace. Learning to support constraint programmers. *Computational Intelligence*, 21(4):336–371, 2005.

[EHM99] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 3(2):124–141, 1999.

[EMS07] N. Eén, A. Mishchenko, and N. Sörensson. Applying logic synthesis for speeding up SAT. In *Theory and Applications of Satisfiability Testing - SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2007.

[ES03a] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[ES03b] A. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.

[FA03] T. Fruewirth and S. Abdennadher. *Essentials of Constraint Programming*. Springer, 2003.

[FD94] D. Frost and R. Dechter. In search of the best constraint satisfaction search. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI'94*, pages 301–306, 1994.

[FDSS08] A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In G. Rudolph et al., editor, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2008.

[FHS11] Álvaro Fialho, Youssef Hamadi, and Marc Schoenauer. Optimizing architectural and structural aspects of buildings towards higher energy efficiency. In Natalio Krasnogor and Pier Luca Lanzi, editors, *GECCO (Companion)*, pages 727–732. ACM, 2011.

[FHS12] Álvaro Fialho, Youssef Hamadi, and Marc Schoenauer. A multi-objective approach to balance buildings construction cost and energy efficiency. In *ECAI*, page to appear, 2012.

[FM02] S. Fitzpatrick and L. Meertens. Scalable, anytime constraint optimization through iterated, peer-to-peer interaction in sparsely-connected networks. In *Proc. IDPT'02*, 2002.

[FS02] S. L. Forman and A. M. Segre. Nagsat: A randomized, complete, parallel solver for 3-sat. In *Proceedings of Theory and Applications of Satisfiability Testing, SAT'02*, pages 236–243, 2002.

[FT63] H. Fisher and L. Thompson. *Industrial Scheduling*, chapter Probabilistic learning combinations of local job-shop scheduling rules. Prentice Hall, 1963.

[Fuk08] A. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.

[Gec06] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.

[GFS08] L. Gil, P. Flores, and L. M. Silveira. PMSat: a parallel version of minisat. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:71–98, 2008.

[GGHM04] C. Gebruers, A. Guerri, B. Hnich, and M. Milano. Making choices using structure at the instance level within a case based reasoning framework. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR*, volume 3011 of *Lecture Notes in Computer Science*, pages 380–386. Springer, 2004.

[GGS07] R. Greenstadt, B. J. Grosz, and M. D. Smith. Ssdpop: improving the privacy of dcop with secret sharing. In *AAMAS*, page 171, 2007.

[GJ08] F. Goualard and C. Jermann. A reinforcement learning approach to interval constraint propagation. *Constraints*, 13(1-2):206–226, 2008.

[GK03] F. Glover and G. Kochenberger. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer, January 2003.

[GL97] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic, Dordrecht, 1997.

[GM04] A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 475–479. IOS Press, 2004.

[Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[Gom03] Carla Gomes. Randomized backtrack search. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology*, pages 233–283. Kluwer, 2003.

[GS97] C.P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proc. UAI'97*, pages 190–197, 1997.

[GS01] C.P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[GS07] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 273–280, Corvalis, Oregon, USA, June 2007. ACM.

[GS08] M. Gagliolo and J. Schmidhuber. Algorithm selection as a bandit problem with unbounded losses. Technical report, Tech. report IDSIA - 07 - 08, 2008.

[GSCK00] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.

[GSK98] Carla Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[GW] Ian Gent and Toby Walsh. CSPLib, a problem library for constraints, `http://www-users.cs.york.ac.uk/~tw/csplib/`.

[GZG+08] Amir Gershman, Roie Zivan, Tal Grinshpoun, Alon Grubstein, and Amnon Meisels. Measuring distributed constraint optimization algorithms. In *AAMAS DCR*, 2008.

[Ham99a] Youssef Hamadi. Optimal distributed arc-consistency. In *Proc. CP'99*, pages 219–233, 1999.

[Ham99b] Youssef Hamadi. *Traitement des problèmes de satisfaction de contraintes distribués*. PhD thesis, Université Montpellier II, 1999. (in french).

[Ham02a] Youssef Hamadi. Distributed, interleaved, parallel and cooperative search in constraint satisfaction networks. Technical Report Technical Report No. HPL-2002-21, HP Laboratories, 2002.

[Ham02b] Youssef Hamadi. Interleaved backtracking in distributed constraint networks. *International Journal on Artificial Intelligence Tools*, 11(2):167–188, 2002.

[Ham02c] Youssef Hamadi. Optimal distributed arc-consistency. *Constraints*, 7(3-4):367–385, 2002.

[Ham03] Youssef Hamadi. Disolver : A Distributed Constraint Solver. Technical Report MSR-TR-2003-91, Microsoft Research, 2003.

[Ham05] Youssef Hamadi. Continuous resources allocation in internet data centers. In *CCGRID*, pages 566–573. IEEE Computer Society, 2005.

[Han08] N. Hansen. Adaptative encoding : How to render search coordinate system invariant. In Springer, editor, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 204–214, 2008.

[HBQ98] Youssef Hamadi, Christian Bessière, and Joël Quinqueton. Distributed intelligent backtracking. In *ECAI*, pages 219–223, 1998.

[HE79] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. In *IJCAI*, pages 356–364, San Francisco, CA, USA, 1979.

[Hen89] P. Van Hentenryck. *Constraint satisfaction in logic programming.* MIT Press, Cambridge, MA, USA, 1989.

[HH93] T. Hogg and B. A. Huberman. Better than the best: The power of cooperation. In *1992 Lectures in Complex Systems*, volume V of *SFI Studies in the Sciences of Complexity*, pages 165–184. Addison-Wesley, 1993.

[HH05] F. Hutter and Y. Hamadi. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK, jan 2005.

[HHHLB06] Frank Hutter, Youssef Hamadi, Holger H. Hoos, and Kevin Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In Frédéric Benhamou, editor, *CP*, pages 213–228, 2006.

[HHS07] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *AAAI*, pages 1152–1157, Vancouver, British Columbia, Canada, July 2007. AAAI Press.

[Hil75] B. Hill. A simple general approach to inference about the tail of a distribution. *Annals of Statistics*, pages 1163–1174, 1975.

[HJS08] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: solver description. Technical Report MSR-TR-2008-83, Microsoft Research, may 2008.

[HJS09] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Control-based clause sharing in parallel SAT solving. In Boutilier [Bou09], pages 499–504.

[HM05] P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.

[HMS08a] Y. Hamadi, E. Monfroy, and F. Saubion. Special issue on autonomous search. *Contraint Programming Letters*, 4, 2008.

[HMS08b] Y. Hamadi, E. Monfroy, and F. Saubion. What is autonomous search? Technical Report MSR-TR-2008-80, Microsoft Research, 2008.

[Hol75] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[Hoo99] H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 296–303. Morgan Kaufmann, 1999.

[Hoo02] H.. Hoos. An adaptive noise mechanism for walksat. In *AAAI/IAAI*, pages 655–660, 2002.

[HQ06] Youssef Hamadi and Claude-Guy Quimper. The smart workflow foundation. Technical Report MSR-TR-2006-114, Microsoft Research, 2006.

[HR06] B. Hu and G. Raidl. Variable neighborhood descent with self-adaptive noeighborhood-ordering. In *Proc. of the 7th EU Meeting on Adaptive, Self-Adaptive and Multilevel Metaheuristics*, 2006.

[HS05] Youssef Hamadi and Marc Shapiro. Pushing log-based reconciliation. *International Journal on Artificial Intelligence Tools*, 14(3):445–458, 2005.

[Hua07] J. Huang. The effect of restarts on the efficiency of clause learning. In M. M. Veloso, editor, *IJCAI*, pages 2318–2323, 2007.

[Hut09] F. Hutter. *Automating the Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, Department of Computer Science, University of British Columbia, 2009.

[HvM06] M. J.H. Heule and H. van Maaren. March dl: Adding adaptive heuristics and a new branching strategy. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:47–59, mar 2006.

[HW09] Shain Haim and Toby Walsh. Restart strategy selection using machine learning techniques. In *SAT*, volume 5584 of *LNCS*, pages 312–325, Swansea, UK, June 2009. Springer.

[HW12] Youssef Hamadi and Christoph M. Wintersteiger. Seven challenges in parallel SAT solving. In *AAAI, Invited paper, to appear*, 2012.

[Ing89] L. Ingber. Very fast simulated re-annealing. *Mathematical Computer Modelling*, 12(8):967–973, 1989.

[Jac88] V. Jacobson. Congestion avoidance and control. In *SIGCOMM*, pages 314–329, 1988.

[JBRS12] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international sat solver competitions. *AI Magazine*, 33(1), 2012.

[JLU05] B. Jurkowiak, C. Min Li, and G. Utard. A parallelization scheme based on work stealing for a class of SAT solvers. *Journal of Automated Reasoning*, 34(1):73–101, 2005.

[JM91] C. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Fourth International Conference on Genetic Algorithms*, pages 31–36, 1991.

[KHR+02] Henry A. Kautz, Eric Horvitz, Yongshao Ruan, Carla P. Gomes, and Bart Selman. Dynamic restart policies. In *AAAI/IAAI*, pages 674–681, 2002.

[Kje91] G. Kjellstroem. On the efficiency of gaussian adaptation. *Journal of Optimization Theory and Applications*, 71(3), 1991.

[Kot10] S. Kottler. SArTagnan: solver description. Technical report, SAT Race 2010, July 2010.

[KP98] S. Kazarlis and V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In *Parallel Problem Solving from Nature - PPSN V, 5th International Conference*, volume 1498 of *Lecture Notes in Computer Science*, pages 211–220, 1998.

[Kra08] O. Kramer. *Self-Adaptive Heuristics for Evolutionary Computation*. Springer, 2008.

[LA97] C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'97*, pages 366–371, 1997.

[Lam78] L. Lamport. Time, clocks and the ordering of events in distributed systems. *Communications of the ACM*, 2:95–104, 1978.

[LB08] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 536–543, Helsinki, Finland, June 2008. ACM.

[LBNA+03] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proc. IJCAI'03*, page 1542, 2003.

[LLM07] F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[LSB07] M. Lewis, T. Schubert, and B. Becker. Multithreaded SAT solving. In *12th Asia and South Pacific Design Automation Conference*, 2007.

[LSZ93] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47:173–180, 1993.

[MFS+09] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, and Michèle Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *IEEE Congress on Evolutionary Computation* [DBL09], pages 365–372.

[MH97] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & OR*, 24(11):1097–1100, 1997.

[Mic92] Z. Michalewicz. *Genetic algorithms + Data structures = Evolution program*. Artificial Intelligence, Berlin: Springer, 1992, 1992.

[MML10] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Improving search space splitting for parallel sat solving. In *ICTAI (1)* [DBL10], pages 336–343.

[MMZ+01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[Mor93] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAIï¿½93)*, pages 40–45. AAAI Press, 1993.

[MS98] K. Marriott and P. Stuckey. *Programming with constraints : an introduction*. MIT Press, Cambridge, Mass., 1998.

[MS08] J. Maturana and F. Saubion. A compass to guide genetic algorithms. In G. Rudolph et al., editor, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2008.

[MSG97] B. Mazure, L. Sais, and E. Grégoire. Tabu search for SAT. In *AAAI/IAAI*, pages 281–285, 1997.

[MSG98] B. Mazure, L. Sais, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Ann. Math. Artif. Intell.*, 22(3-4):319–331, 1998.

[MSL04] E. Monfroy, F. Saubion, and T. Lambert. On hybridization of local search and constraint propagation. In *Logic Programming, 20th International Conference, ICLP 2004*, volume 3132 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2004.

[MSS96] J. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[MSTY05] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161, 2005.

[NE06] V. Nannen and A. E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings*, pages 183–190. ACM, 2006.

[NE07] V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 975–980, 2007.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(). *J. ACM*, 53(6):937–977, 2006.

[NSE08] V. Nannen, S. Smit, and A. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In *Parallel Problem Solving from Nature - PPSN X, 10th International Conference*, volume 5199 of *Lecture Notes in Computer Science*, pages 528–538. Springer, 2008.

[OGD06] L. Otten, M. Grönkvist, and D. P. Dubhashi. Randomization in constraint programming for airline planning. In *CP*, pages 406–420, 2006.

[OHH+08] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *AICS*, Aug 2008.

[PD07] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In J. Marques-Silva and K. A. Sakallah, editors, *Theory and Applications of Satisfiability Testing SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.

[PH06] Wayne J. Pullan and Holger H. Hoos. Dynamic local search for the maximum clique problem. *J. Artif. Intell. Res. (JAIR)*, 25:159–185, 2006.

[PHS08] Cédric Piette, Youssef Hamadi, and Lakhdar Sais. Vivifying propositional clausal formulae. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 525–529. IOS Press, 2008.

[PHS09] Cédric Piette, Youssef Hamadi, and Lakhdar Sais. Efficient combination of decision procedures for mus computation. In Silvio Ghilardi and Roberto Sebastiani, editors, *FroCos*, volume 5749 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2009.

[PK01] D. Patterson and H. Kautz. Auto-walksat: A self-tuning implementation of walksat. *Electronic Notes in Discrete Mathematics*, 9:360–368, 2001.

[PR08] J. Puchinger and G. Raidl. Bringing order into the neighborhoods: relaxation guided variable neighborhood search. *J. Heuristics*, 14(5):457–472, 2008.

[RBea05] I. Rish, M. Brodie, and S. Ma et al. Adaptive diagnosis in distributed dystems. *IEEE Trans. on Neural Networks*, 16:1088–1109, 2005.

[RBHB06] Carsten Rother, Lucas Bordeaux, Youssef Hamadi, and Andrew Blake. Autocollage. *ACM Trans. Graph.*, 25(3):847–852, 2006.

[Ref04] Philippe Refalo. Impact-based search strategies for constraint programming. In Mark Wallace, editor, *CP*, volume 3258 of *LNCS*, pages 557–571, Toronto, Canada, Sept 2004. Springer.

[RH05] Georg Ringwelski and Youssef Hamadi. Boosting distributed constraint satisfaction. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 549–562. Springer, 2005.

[Ric75] John R. Rice. The algorithm selection problem. Technical Report CSD-TR 152, Computer science department, Purdue University, 1975.

[Ric76] John R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118, 1976.

[Rin11] J. Rintanen. Heuristics for planning with SAT and expressive action definitions. In F. Bacchus, C. Domshlak, S. Edelkamp, and M. Helmert, editors, *ICAPS*. AAAI, 2011.

[RS08] V. Ryvchin and O. Strichman. Local restarts. In Büning and Zhao [BZ08], pages 271–276.

[SBCH04] Marc Shapiro, Karthikeyan Bhargavan, Yek Chong, and Youssef Hamadi. A formalism for consistency and partial replication. Rapport technique, 2004.

[SBH07] Sathiamoorthy Subbarayan, Lucas Bordeaux, and Youssef Hamadi. Knowledge compilation properties of tree-of-bdds. In *AAAI*, pages 502–507. AAAI Press, 2007.

[SE09] Selmar K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *IEEE Congress on Evolutionary Computation* [DBL09], pages 399–406.

[SF05] M.-C. Silaghi and B Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence*, 161, 2005.

[SGS07] Matthew Streeter, Daniel Golovin, and Stephen F. Smith. Combining multiple heuristics online. In *AAAI*, pages 1197–1203, Vancouver, British Columbia, Canada, July 2007. AAAI Press.

[SKC94] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *AAAI*, pages 337–343, 1994.

[SLB10] T. Schubert, M. Lewis, and B. Becker. Antom: solver description. Technical report, SAT Race, 2010.

[SLS+08] Marius Silaghi, Rob Lass, Evan Sultanik, William Regli, Toshihiro Matsui, and Makoto Yokoo. The operation point units of distributed constraint solvers. In *AAMAS DCR*, 2008.

[SM07] Horst Samulowitz and Roland Memisevic. Learning to solve QBF. In *AAAI*, pages 255–260, Vancouver, British Columbia, July 2007. AAAI Press.

[SM08] K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25, 2008.

[SSHF00] Marius-Calin Silaghi, Djamila Sam-Haroud, and Boi Faltings. Asynchronous search with aggregations. In *Proc. AAAI/IAAI 2000*, pages 917–922, 2000.

[Syw89] G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Thi05] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In H.-G. Beyer, editor, *Proc. GECCO'05*, pages 1539–1546. ACM Press, 2005.

[Thi07] D. Thierens. Adaptive Strategies for Operator Allocation. In F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 77–90. Springer Verlag, 2007.

[Tho00] J. Thornton. *Constraint Weighting for Constraint Satisfaction*. PhD thesis, School of Computing and Information Technology, Griffith University, Brisbane, Australia, 2000.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1^{st} edition, August 1993.

[Vap95] Vladimir Vapnik. *The Nature of Statistical Learning*. Springer Verlag, 1995.

[Wal00] T. Walsh. SAT v CSP. In *Proc. of CP 2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[WB08] Huayue Wu and Peter Van Beek. Portfolios with deadlines for backtracking search. In *IJAIT*, volume 17, pages 835–856, 2008.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining - Practical Machine Learning Tools and Techniques*. Elsevier, 2005.

[WHdM09] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. A concurrent portfolio approach to SMT solving. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 715–720. Springer, 2009.

[WHdM10] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. Efficiently solving quantified bit-vector formulas. In Roderick Bloem and Natasha Sharygina, editors, *FMCAD*, pages 239–246. IEEE, 2010.

[WLLH03] Wong, Lee, Leung, and Ho. A novel approach in parameter adaptation and diversity maintenance for GAs. *Soft Computing*, 7(8):506–515, 2003.

[WPS06a] J. Whitacre, Q. Tuan Pham, and R. Sarker. Credit assignment in adaptive evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2006*, pages 1353–1360. ACM, 2006.

[WPS06b] J. Whitacre, T. Pham, and R. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1345–1352. ACM, 2006.

[XHHLB07] L. Xu, F Hutter, H. Hoos, and K. Leyton-Brown. Satzilla-07: The design and analysis of an algorithm portfolio for SAT. In *Principles and Practice of Constraint Programming - CP 2007*, 2007.

[XHHLB08] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[YD98] M. Yokoo and E. H. Durfee. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 1998.

[YDIK92] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proc. ICDCS'92*, pages 614–621, 1992.

[YG04] B. Yuan and M. Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In X. Yao et al., editor, *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference*, volume 3242 of *Lecture Notes in Computer Science*, pages 172–181. Springer, 2004.

[YG05] F. Yu-Hui Yeh and M. Gallagher. An empirical study of hoeffding racing for model selection in k-nearest neighbor classification. In M. Gallagher, J. Hogan, and F. Maire, editors, *IDEAL*, volume 3578 of *Lecture Notes in Computer Science*, pages 220–227. Springer, 2005.

[YG07] B. Yuan and M. Gallagher. Combining meta-eas and racing for difficult EA parameter tuning tasks. In Lobo et al. [LLM07], pages 121–142.

[ZBH96] H. Zhang, M. P. Bonacina, and J. Hsiang. Psato: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21:543–560, 1996.

[ZM03] R. Zivan and A. Meisels. Synchronous vs asynchronous search on DisCSPs. In *Proc. EUMAS'03*, 2003.

[ZM05] Roie Zivan and Amnon Meisels. Concurrent search for distributed csps. *Artificial Intelligence*, 161, 2005.

[ZMMM01] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.

[ZS94] H. Zhang and M. E. Stickel. Implementing the davis-putnam algorithm by tries. Technical report, Artificial Intelligence Center, SRI International, Menlo, 1994.

Part II

Curriculum Vitae

Professional Appointments

2009 – *present*	Founder, co-Director, Microsoft-CNRS chair Optimization and Sustainable Development at École Polytechnique, Palaiseau, France.
2009 – *present*	Associate Researcher, LIX Laboratory, École Polytechnique, Palaiseau, France.
2007 – *present*	Project co-Leader, Adaptive Combinatorial Search for e-Sciences, MSR/INRIA joint-lab, Orsay, France.
2003 – *present*	Senior Researcher (Directeur de Recherche), Head of Constraint-Reasoning, Microsoft Research, Cambridge, United Kingdom.
2001 – 2003	Research Scientist, Hewlett-Packard Laboratory, Bristol, United Kingdom.
2000 – 2001	Research Scientist, *Laboratoire Central de Recherche*, Thales-Research, Orsay, France.
1998 – 1999	Associate Lecturer, University of Montpellier, France.

Education

2009	General Management Certificate (selected MBA courses), Executive Education, Judge Business School, University of Cambridge, United Kingdom.
1999	Ph.D. in Computer Science, University of Montpellier, (Summa Cum Laude) France.
1995	Master in Computer Science, University of Montpellier, France.
1993	Bachelor in Mathematics and Computer Sciences, University of Montpellier, France.
	Bachelor in Applied Mathematics (MASS), University of Montpellier, France

Board Memberships

2012	Intelligent Systems Applications Technical Committee of the IEEE Computational Intelligence Society.
2011 – *2014*	Institut National de Recherche en Agronomie, Mathematics and Applied Computer Sciences department (INRA-MIA), Scientific board.
2009 – *present*	*Reactive Search SrL*, Scientific advisor.
2004 – *present*	University of Saint-Andrews, School of Computer Science, Industrial Board.

Awards

2011

- SAT-Competition 2011: Silver medal for the CSLS solver in the SAT+Random category.

2010

- ROADEF/EURO Challenge 2010: A large-scale energy management problem with varied constraints, supervision for winning team senior category, Lisbon, Portugal.
- SAT-Race 2010: Silver and Bronze medals for the ManySAT solver, Edinburgh, Scotland.

2009

- IEEE Ramamoorthy Best Paper Award, "Learning for Dynamic Subsumption", International Conference on Tools for Artificial Intelligence (ICTAI'09), Newark, USA.
- SAT-Competition 2009: Gold, Silver and Bronze medals for the ManySAT solver with title of « Best Parallel SAT Solver », Swansea, United Kingdom.
- *Microsoft Gold Star Award* for the definition of the "Optimization for Sustainable Development" project.

2008

- SAT-Race 2008 : Gold medal for the ManySAT solver with title of « Best Parallel SAT Solver », http://baldur.iti.uka.de/sat-race-2008/, Guangzhou, P. R. China.
- *Microsoft Gold Star Award* for the work on the Disolver constraint solver and its integration into Microsoft Office (MS-Project).

Press

2011

- EDF R&D Magazine, « Un challenge pour mieux gérer le parc thermique », number 1, July 2011.

2009

- Futures Microsoft's European Innovation Magazine, « The Marriage of Maths and Ecology », number 5, December 2009.
- Interview, *La Tribune*, "Microsoft, l'X et le CNRS font chaire commune", June 5th.
- Featured in *Le Monde Informatique*, " Microsoft, le CNRS et Polytechnique créent une chaire « Optimisation et Développement durable»", June 9th.
- Featured in *Le Journal du Net*, "Microsoft et le CNRS créent une chaire commune" , June 4th.
- Featured in *Silicon.fr,* "Microsoft Solver Foundation", February 15th.
- Microsoft *Futures* Magazine, "No Constraints", number 4, June 2009.
- Featured in *Le Monde Informatique,* "Le labo commun Microsoft-Inria expose ses projets", January 29th.

2008

- *Regards sur le numérique*, Portrait, Youssef Hamadi : Sans Contraintes, numéro 3, June 2008.
- Feverish Foray into Satisfiability Proves Satisfying, Microsoft feature-story.

Systems and Software

2009

- Parallel algorithms for the Z3 2.0, Satisfiability Modulo Theory Solver, http://research.microsoft.com/en-us/um/redmond/projects/z3/.
- Microsoft Solver Foundation 2.0, Constraint Programming and Local Search components. http://www.solverfoundation.com.

2008

- Microsoft Solver Foundation 1.0, Constraint Programming and Local Search components (Disolver-based). http://www.solverfoundation.com.

2006

- Microsoft Office Project, Disolver-based Portfolio optimization component. http://office.microsoft.com/en-gb/project/default.aspx.
- Microsoft Research's AutoCollage graphics tool, http://research.microsoft.com/en-us/um/cambridge/projects/autocollage/.

2005

- Scalable, Automated, Guided Execution (SAGE) Disolver-based engine for automated software verification.

2003

- Disolver, the distributed constraint solver, v1.0.

2001

- Thales Research, in charge of the online scheduling algorithm for the Principal Anti-Air Missile System (PAAMS). http://en.wikipedia.org/wiki/PAAMS.

Recent Invited Keynotes, Lectures, and Courses

2011

- *On Microsoft Research Policies*, Towards a Global Observatory of Policy Instruments on Science, Technology and Innovation UNESCO Workshop, Paris October 19-20th 2011.
- *SmartBuildings*, Green Growth Leaders Workshop, Copenhagen, October 12th 2011.
- *Approaches to Parallel SAT*, lecturer first MIT Summer School on SAT/SMT, Cambridge (MA), June 2011.
- *Parallel SAT*, Instituto de Engenharia de Sistemas e Computadores Investigação (INESC-ID), Lisbon, Portugal, February 2011.

2010

- *The Smart Workflow Foundation*, Cambridge Network Special Interest Groups on Business Intelligence, Anglia Business Solutions, Cambridge, UK.
- *An Overview of Parallel SAT solving*, 22th International Conference on Tools with Artificial Intelligence (ICTAI'10), Arras, France, October 2010.
- *Parallel SAT Solving*, 9th International Workshop on Parallel and Distributed Methods in verification (PDMC'10), Twente, The Netherlands, October 2010.
- *Recent Improvements in Parallel SAT Solving*, Federated Logic Conference, tutorial, Edinburgh, Scotland, POS, July 2010.

- *Decision Procedures in Software, Hardware and Bioware*, Schloss Dagstuhl Seminar 10161, Germany, April 2010.
- *Constraints in Microsoft*, Industrial Talks for Constraint Satisfaction Module, University of Essex.
- *On SAT and Parallel SAT*, École Polytechnique, cours INF580, March 2010.

2009

- *Control-based clause-sharing in Parallel SAT***,** CS seminar, University of Caen, December 2009.
- *Algorithms and Applications for Next Generation SAT Solvers*, Schloss Dagstuhl Seminar 09461, Germany, November 2009.
- *From SAT to efficient Parallel SAT solving***,** École Polytechnique, (cours INF580) France.
- *From SAT to efficient Parallel SAT solving***,** Oxford University, CS seminar.
- *From SAT to efficient Parallel SAT solving***,** Tutorial, Learning for Intelligent Optimization (LION'09), University of Trento, Italy.
- *Control-based clause-sharing in Parallel SAT***,** Conference on "Complex Networks across the Natural and Technological Sciences", Institute for Advanced Studies, University of Strathclyde, UK.
- *Control-based clause-sharing in Parallel SAT***,** Thales Research, seminar, Paris, France.

2007

- *Constraint Programming and Satisfiability: a Comparative Survey*, Tutorial, Summer School on Decision Aiding (COSI'07), Oran, Algeria.

Editorial Responsibilities

2012

- **Conference co-Chair** Learning for Intelligent Optimization (LION), Paris, January 16-20 2012.
- **Member of Program Committee**, Formal Methods in Computer-Aided Design (FMCAD), 15th International Conference on Theory and Applications of Satisfiability Testing (SAT), International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR), Learning and Intelligent Optimization Conference (LION), Colloque sur l'Optimisation et les Systèmes d'Information (COSI), Workshop on Practical and Distributed Model Checking (PDMC).
- **External Reviewing**, Constraints Journal, Journal of Artificial Intelligence Research (JAIR).

2011

- **co-Editor** Book "Autonomous Search", ISBN 978-3-642-21433-2, Springer 2011.
- **Member of Program Committee**, 14th International Conference on Theory and Applications of Satisfiability Testing (SAT), International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR), Learning and Intelligent Optimization Conference (LION), Colloque sur l'Optimisation et les Systèmes d'Information (COSI), High Performance Computing Symposium (HPCS) Workshop on Parallel SAT, 3rd International workshop on Constraints in Software Testing, Verification and Analysis (CSTVA), Workshop on Practical and

Distributed Model Checking (PDMC), International Conference on Tools for Artificial Intelligence (ICTAI),), Journées Francophones de Programmation par Contraintes (JFPC).

- **External Reviewing**, Constraints Journal, IEEE Transactions on Parallel and Distributed Systems, Journal of Artificial Intelligence Research (JAIR).

2010

- **Member of Advisory Panel**, SAT Race 2010.
- **Member of Program Committee**, International Conference on Artificial Intelligence (AAAI), International Conference on Artificial Intelligence (AAAI) Nectar Track, International Conference on Principles and Practice of Constraint Programming (CP), Learning and Intelligent Optimization Conference (LION), International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR), Colloque sur l'Optimisation et les Systèmes d'Information (COSI), High Performance Computing Symposium (HPCS) Workshop on Parallel SAT, Second International workshop on Constraints in Software Testing, Verification and Analysis (CSTVA), 3rd Workshop on Techniques for Implementing Constraint Programming Systems, 3rd Workshop on Constraint Reasoning and Optimization for Computational Sustainability.
- **External Reviewing**, INFORMS Journal on Computing, ACM Transaction on Intelligent Systems and Technology, Constraints Journal.

2009

- **Tutorials Chair**, Learning and Intelligent Optimization Conference (LION).
- **Member of Program Committee**, International Joint Conference on Artificial Intelligence (IJCAI), Learning and Intelligent Optimization Conference (LION), International Symposium on Combinatorial Search (SoCS), Journées Francophones de Programmation par Contraintes (JFPC), Distributed Constraint Reasoning (DCR) Workshop, High Performance Computing Symposium (HPCS) Workshop on Parallel SAT.
- **Guest Editor**, Journal on Satisfiability, Boolean Modeling and Computation, Volume 6 Special Issue on “Parallel SAT Solving”.
- **Workshop co-organizer**: First International Workshop on Constraint Reasoning and Optimization for Computational Sustainability (CP’09).

2008

- **Guest co-Editor,** Constraint Programming Letters, Special Issue on “Autonomous Search”.
- **Member of Program Committee**, International Conference on Artificial Intelligence (AAAI), International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR), IEEE International Symposium on Cluster, Cloud, and Grid Computing (ccGrid), Learning and Intelligent Optimization Conference (LION), Journées Francophones de Programmation par Contraintes (JFPC), Colloque sur l'Optimisation et les Systèmes d'Information (COSI), Search in AI and Robotics Workshop (AAAI’08), Distributed Constraint Reasoning (DCR) Workshop (AAMAS’08), CONFIG Workshop (ECAI’08).
- **Workshop organizer**: First Int. Workshop on “Search in ManyCore” (CP’08).
- **External Reviewing**, INFORMS Journal on Computing, Journal on Satisfiability, Boolean Modeling and Computation.

2007

- **Guest Editor**, Journal on Satisfiability, Boolean Modeling and Computation, Special Issue on “SAT/CP Integration”.
- **Member of Program Committee**, IEEE International Symposium on Cluster, Cloud, and Grid Computing (ccGrid), Journées Francophones de Programmation par Contraintes (JFPC), Learning and Intelligent Optimization Conference (LION), Colloque sur l'Optimisation et les Systèmes d'Information (COSI), Learning for Search Workshop (AAAI’07), Distributed Constraint Reasoning (DCR) Workshop (CP’07), CONFIG Workshop (AAAI’07).
- **Workshop co-organizer**: First Int. Workshop on Autonomous Search (CP’07).
- **External Reviewing**, Journal on Satisfiability, Boolean Modeling and Computation (JSAT), Journal of Artificial Intelligence Research (JAIR).

2006

- **Member of Program Committee**, IEEE International Symposium on Cluster, Cloud, and Grid Computing (ccGrid), Journées Francophones de Programmation par Contraintes (JFPC), Colloque sur l'Optimisation et les Systèmes d'Information (COSI).
- **Workshop organizer**: First Int. Workshop on SAT/CP Integration (CP’06).

2005

- **Member of Program Committee**, IEEE International Symposium on Cluster, Cloud, and Grid Computing (ccGrid), Journées Francophones de Programmation par Contraintes (JFPC), Learning and Intelligent Optimization Conference (LION), Colloque sur l'Optimisation et les Systèmes d'Information (COSI), Distributed Constraint Reasoning (DCR) Workshop.

2004

- **Member of Program Committee**, Distributed Constraint Reasoning (DCR) Workshop.

PhD Committee Membership

2011	Alejandro Arbelaez, Université Paris-Sud/INRIA-MSR, France.
	Ruben Martins, INESC-ID Lisboa, Portugal.
2010	Alvaro Fialho, LRI, Université Paris-Sud, France.
2008	Fabien Corblin, IMAG-LSR, Université Joseph Fourier, France.
	Said Jabbour, Université d’Artois CRIL/CNRS, France.
2007	Cédric Piette, Université d’Artois CRIL/CNRS, France.

PhD co-Supervision

2011 – *present*	Van-Anh Nguyen, University of Caen, GREYC, France.
2009 – *present*	Long Guo, Université d’Artois. CRIL-CNRS, France.
2009 – *present*	Nadarajen Veerapen, Université d’Angers, France.
2008 – *present*	Nicolas Mobilia, TIMC-IMAG, Grenoble, France
2007 – *2011*	Alejandro Arbaleaz, Université Paris-Sud/INRIA-MSR, France.
2006 – 2008	Fabien Corblin, IMAG-LSR, Université Joseph Fourier France.

PhD Summer Internship Supervisions

2012 Walid Trabelsi, University College Cork, Ireland.
2010 Mauro Bampo, Monash University, Australia.
2009 Said Jabbour, Université d'Artois, Lens, France.
2008 Dejan Vojanovic, New-York University, USA.
2007 Fabien Corblin, TIMC-IMAG, Grenoble, France.
2007 Cedric Piette, Université d'Artois, Lens, France.
2006 Claude-Guy Quimper, Université Laval, Québec, Canada.
2006 Frank Hutter, University of British Colombia, Canada.
2005 Sathiamoorthy Subbarayan, ITU, Denmark.
2004 Georg Ringwelski, Cork Constraint Computation Centre, Ireland.

Patents

1 **Ordering decision nodes in distributed decision making**, Youssef Hamadi, Yek Loong Chong, Marc Shapiro. US Patent No. 7222149, May 2007.
2 **Multi-Ordered Distributed Constraint Search**, Youssef Hamadi, and Georg Ringwelski, US Patent No. 7472094, December 2008.
3 **Auto Collage,** Carsten Rother, Lucas Bordeaux, Youssef Hamadi, Andrew Blake, US Patent No. 7529429, May 2009.
4 **Tuning of Problem Solvers**, Youssef Hamadi, US Patent No. 7890439, February 2011.
5 **Graphical Acquisition of an Objective Function**, Youssef Hamadi and Stéphane Ubeda, US Patent No. 7975234, July 2011.
6 **Generation of commercial presentations**, Youssef Hamadi, and Carsten Rother, US Patent App. No. 11465335, August 2006.
7 **Constrained Exploration for Search Algorithms**, Youssef Hamadi, US Patent App. No. 11170290, June 2005.
8 **Resource balancing in distributed peer to peer networks**, Youssef Hamadi, US Patent App. No. 10891262, July 2004.
9 **Adapting computer resource usage based on forecasted resource availability**, Youssef Hamadi, US Patent App. No. 10931818, September 2004.
10 **Context Sensitive Camera (RFID-based)**, Youssef Hamadi, US Patent App. No. 10659121, September 2003.
11 **Online recognition of robots**, Youssef Hamadi, Maher Rahmouni, US Patent App. No. 10421301, April 2003.
12 **Configuration of computer networks**, Youssef Hamadi, US Patent App. No. 10281923, October 2002.
13 **Method for constraining file systems in peer to peer networks**, Youssef Hamadi, US Patent App. No. 10269359, October 2002.
14 **Method for designing optimization algorithms integrating a time limit,** Youssef Hamadi, Simon De Givry, US. Patent App. 10496513, November 2002.
15 **Robustness of a Workflow**, Youssef Hamadi and Claude-Guy Quimper, US Patent App. No. 11669064.

16 **Allocating Resources to Tasks in Workflows**, Youssef Hamadi and Claude-Guy Quimper, US Patent App. No. 11669098.
17 **Synchronizing Workflow**, Youssef Hamadi and Claude-Guy Quimper, US Patent App. No. 11669082.
18 **Controlled Constraint Sharing in Parallel Problem Solvers**, Youssef Hamadi and Said Jabbour, US Patent App. No. 12465440.

Peer-reviewed publications

Books

1. Autonomous Search, Y. Hamadi, F. Saubion, and E. Monfroy, edited book, ISBN 978-3-642-21433-2, Springer 2012.
2. Autonomous Search, Y. Hamadi, E. Monfroy, and F. Saubion, invited book chapter in "CPAIOR 10th anniversary" Eds. M. Milano and P. Van Hentenryck, Springer 2010.
3. Connection and Integration with SAT Solvers: A Survey and a Case Study with the Problem of Deciphering Discrete Genetic Networks, F. Corblin, E. Fanchon, L. Bordeaux, Y. Hamadi, and L. Trilling, invited book chapter in "CPAIOR 10th anniversary" Eds. M. Milano and P. Van Hentenryck, Springer 2010.
4. On the First SAT/CP Integration Workshop, L. Bordeaux, Y. Hamadi, invited book chapter in "Trends in Constraint Programming", p105-123, ISTE, 2007.
5. Problèmes de satisfaction de contraintes et systèmes Multi-agents, Y. Hamadi, and S. Piechviak, invited book chapter in "Systèmes Multi-agents" Ed. Editor R. Mandieu. Hermès p169-205, 2002.

Journals

1. Efficiently solving quantified bit-vector formulas, C. M. Wintersteiger, Y. Hamadi, and L. de Moura, Formal Methods in System Design (**FMSD**) Invited article, Special issue 10 years to the SMT initiative, 2012.
2. Challenges in Parallel SAT Solving, Y. Hamadi, and C. M. Wintersteiger, Invited article, AI Magazine (**AI-Mag**), to appear.
3. Learning from Conflicts in Propositional Satisfiability, Y. Hamadi, S. Jabbour, and L. Sais, Invited Survey, 4OR: A Quarterly Journal of Operations Research (**4OR**), 10(1), 2012.
4. Deterministic Parallel DPLL: System Description, Y. Hamadi, S. Jabbour, C. Piette, and L. Sais, Int. Journal on Satisfiability, Boolean Modeling and Computation (**JSAT**), Volume 7, 2011.
5. Boosting Distributed Constraint Satisfaction, Y. Hamadi, and G. Ringwelski, **Journal of Heuristics** 2011, Volume 17, Number 3, 251-279
6. Learning for Dynamic Subsumption, Y. Hamadi, S. Jabbour, and L. Sais, invited in Special Issue of the International Journal on Artificial Intelligence Tools (**IJAIT**) vol 19 no 4, August 2010.
7. ManySAT: a Parallel SAT Solver, Y. Hamadi, S. Jabbour, and L. Sais, Int. Journal on Satisfiability, Boolean Modeling and Computation (**JSAT**), Volume 6, Special Issue on Parallel SAT, Ed. Y. Hamadi, IOS Press, 2009.
8. Conclusion to the Special Issue on Parallel SAT Solving, Editor Y. Hamadi, Int. Journal on Satisfiability, Boolean Modeling and Computation (**JSAT**), Volume 6, IOS Press 2009.
9. Editor Special Issue on Autonomous Search, Y. Hamadi, E. Montfroy, and F. Saubion, Constraint Programming Letters (**CPL**), Volume 4, 2008.

10. Editor Special Issue on SAT/CP Integration, L. Bordeaux, Y. Hamadi, Int. Journal on Satisfiability, Boolean Modeling and Computation (**JSAT**), IOS Press, 2007.
11. Propositional Satisfiability and Constraint Programming: A comparative Survey L. Bordeaux, Y. Hamadi, and L. Zhang, ACM Computing Surveys (**CSUR**), Volume 38, Issue 4 (2006).
12. AutoCollage, C. Rother, L. Bordeaux, Y. Hamadi, and A. Blake, (**SIGGRAPH'06**) ACM Trans. Graph. 25(3): 847-852 (2006).
13. Conflicting Agents in Distributed Search, Y. Hamadi,in International Journal on Artificial Intelligence Tools (**IJAIT**), N. G. Bourbakis editor-in-chief, Volume 14, Number 3-4, 2005. (Invited Paper.)
14. Pushing log-based Reconciliation, Y. Hamadi and M. Shapiro,in International Journal on Artificial Intelligence Tools (**IJAIT**), N. G. Bourbakis editor-in-chief, Volume 14, Number 3-4, 2005.
15. Interleaved search in distributed constraint networks, Y. Hamadi, in International Journal on Artificial Intelligence Tools (**IJAIT**), N. G. Bourbakis editor-in-chief, Volume 11, Number 4, p167-188, 2002.
16. Optimal Distributed Arc-Consistency, Y. Hamadi, in **Constraints** Journal, Invited paper for the special Issue on 1998 and 1999 International Conference on Principles and Practice of Constraint Programming. Editors Joxan Jaffar and Michael J. Maher. Kluwer Academic Publishers vol. 7, No. 3/4, July/October 2002.
17. Architectures reconfigurables et traitement de problèmes NP-difficiles : un nouveau domaine d'application, Y. Hamadi, D. Merceron, Revue Techniques et Sciences Informatiques, (**TSI**) numéro spécial Architectures reconfigurables, vol. 18 n. 10, p1113-1135, Editions Hermès 1999.

Conferences

1. Seven Challenges in Parallel SAT Solving, Y. Hamadi, and C. M. Wintersteiger, Invited paper, Twenty-Sixth AAAI Conference (**AAAI-12**), to appear.
2. A Multi-objective Approach to Balance Buildings Construction Cost and Energy Efficiency, A Fialho, Y. Hamadi, and M. Schoenauer, Twentieth European Conference on Artificial Intelligence (**ECAI'12**), Montpellier, to appear.
3. Improving Parallel Local Search for SAT, A. Arbelaez, Y. Hamadi, Learning and Intelligent Optimization (**LION'11**), Roma, Italy.
4. Efficiently Solving Quantified Bit-Vector Formulas, C. Wintersteiger, Y. Hamadi, and L. de Moura, Formal Methods in Computer Aided Design (**FMCAD'10**), Lugano, Switzerland.
5. Adaptive K-Parallel Best-First Search: A Simple but Efficient Algorithm for Multi-Core Domain-Independent Planning, V. Vidal, L. Bordeaux, and Y. Hamadi, Third International Symposium on Combinatorial Search (**SoCS'10**), July 2010, Stone Mountain, Atlanta, USA.
6. Diversification and Intensification in Parallel SAT Solving, L. Guo, Y. Hamadi, S. Jabbour, and L. Sais, 16th International Conference on Principles and Practice of Constraint Programming (**CP'10**).
7. Continuous Search in Constraint Programming, A. Arbelaez, Y. Hamadi, and M. Sebag, Twenty-second International Conference on Tools with Artificial Intelligence (**ICTAI'10**), October 2010, Arras, France.
8. Control-based Clause Sharing in Parallel SAT Solving, Y. Hamadi, S. Jabbour, and L. Sais, Twenty-first International Joint Conference on Artificial Intelligence (**IJCAI'09**), July 2009, Pasadena, USA.

9. Experiments with Massively Parallel Constraint Solving, L. Bordeaux, Y. Hamadi, and H. Samulowitz, Twenty-first International Joint Conference on Artificial Intelligence (**IJCAI'09**), July 2009, Pasadena, USA.
10. A Concurrent Portfolio Approach to SMT Solving, C. Wintersteiger, Y. Hamadi, and L. de Moura, Twenty-one International Conference on Computer Verification (**CAV'09**), June 2009, Grenoble, France.
11. A new collaborative scheme for computing a MUS, C. Piette, Y. Hamadi, and L. Sais, Seventh International Symposium on Frontiers of Combining Systems (**FroCos'09**), September 2009, Trento, Italy.
12. Learning for Dynamic Subsumption, Y. Hamadi, S. Jabbour, and L. Sais, Twenty-first International Conference on Tools with Artificial Intelligence (**ICTAI'09**), November 2009, Pasadena, USA. (Best Paper Award.)
13. Continuous Search in Constraint Programming: An Initial Investigation, A. Arbelaez, and Y. Hamadi, The Fifteen International Conference on Principles and Practice of Constraint Programming (**CP'09**), doctoral program, September 2009, Lisbon, Portugal.
14. Online Heuristic Selection in Constraint Programming, A. Arbelaez, Y. Hamadi, and M. Sebag, International Symposium on Combinatorial Search (**SoCS'09**), July 2009, Lake Arrowhead, USA.
15. Subsumption dynamique basée sur l'apprentissage, Y. Hamadi, S. Jabbour, and L. Sais, Journées Francophones de Programmation par Contraintes (**JFPC'09**), Juin 2009.
16. Réordonnancement dynamique basé sur l'apprentissage, Y. Hamadi, S. Jabbour, and L. Sais, Journées Francophones de Programmation par Contraintes (**JFPC'09**), Juin 2009.
17. Exploiting Weak Dependencies in Tree-based Search, A. Arbelaez, and Y. Hamadi, 24th Annual ACM Symposium on Applied Computing (**SAC'09**), March 2009 Hawaii, USA.
18. Vivifying Propositional Clausal Formulae, C. Piette, Y. Hamadi, and L. Sais, Eighteen European Conference on Artificial Intelligence (**ECAI'08**), July 2008, Patras, Greece.
19. A Generalized Framework for Conflict Analysis, G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais, (**SAT'08**).
20. Un cadre général pour l'analyse de conflits, G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais, Journées Francophones de Programmation par Contraintes (**JFPC'08**), Juin 2008.
21. Vivification de formules propositionnelles clausales, C. Piette, Y. Hamadi, and L. Sais, Journées Francophones de Programmation par Contraintes (**JFPC'08**), Juin 2008.
22. Knowledge Compilation Properties of Trees-of-BDDs, S. Subbarayan, L. Bordeaux, and Y. Hamadi, (**AAAI'07**), p502-507.
23. An analysis of Slow Convergence in Interval Propagation, L. Bordeaux, Y. Hamadi, and M. Vardi, (**CP'07**), p790-797.
24. Iterated Expressions in Constraint Programming, L. Bordeaux, Y. Hamadi, C-G Quimper, and H. Samulowitz, Journées Francophones de Programmation par Contraintes (**JFPC'07**), Juin 2007.
25. Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms, F. Hutter, Y. Hamadi, H. Hoos and K. L. Brown, Twelfth International Conference on Principles and Practice of Constraint Programming (**CP'06**).
26. Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms: An Initial Investigation, F. Hutter, Y. Hamadi, H. Hoos and K. L. Brown, (**AAAI'06**), First International Workshop on Learning for Search.
27. Distributed Log-based Reconciliation, Y. Chong and Y. Hamadi, Seventeen European Conference on Artificial Intelligence (**ECAI'06**), August 2006, Riva del Garda, Italy.
28. Boosting Distributed Constraint Satisfaction, G. Ringwelski and Y. Hamadi, Eleventh International Conference on Principles and Practice of Constraint Programming (**CP'05**).

29. An Overview of Sangam: A System for Integrating Data to Investigate Stress-Circuitry-Gene Coupling, M. Saxena, S. Kim, G. Burns, A. M. Khan, J. Su, Y. Hamadi and S. Ghandeharizadeh, (**IVNET'05**).
30. Continuous Resources Allocation in Internet Data Centers, Y. Hamadi, 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (**ccGRID'05**) , 9-12 May 2005, Cardiff, United Kingdom.
31. The Temporal Knapsack Problem and its Solution, M. Bartlett,A. M. Frisch, Y. Hamadi, I. Miguel, S. A.Tarimand C. Unsworth, Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (**CP-AI-OR'05**) 29-1 June 2005.
32. A formalism for consistency and partial replication, K. Bhargavan, M. Shapiro, Y. Chong and Y. Hamadi, in the 18th Annual Conference on Distributed Computing (**DISC'04**), Trippenhuis, Amsterdam, the Netherlands, October 4-7.
33. Interleaved Backtracking in Distributed Constraint Networks, 13th International Conference on Tools with Artificial Intelligence (**ICTAI'01**), IEEE, p33-41, October 2001.
34. Optimal Distributed Arc-Consistency, Y. Hamadi, Fifth International Conference on Principles and Practice of Constraint Programming (**CP'99**), Alexandria USA, Lecture Notes in Computer Science Vol. 1713, p219-233, Springer, October 1999.
35. Communication et Emergence : une épidémie chez les termites J. Quinqueton et Y. Hamadi, Actes des Journées Francophones d'Intelligence Artificielle Distribuée et de Système Multi-Agents (**JFIADSMA'99**), La Réunion, France, Editions Hermès p225-235.
36. Backtracking in Distributed Constraint Networks, Y. Hamadi, C. Bessière and J. Quinqueton , Thirteen European Conference on Artificial Intelligence (**ECAI'98**), Brighton, England, p219-223.
37. Reconfigurable architectures: A new vision for optimization problems, Y. Hamadi, D. Merceron, Third International Conference of Principles and Practice of Constraint Programming (**CP97**), Linz Austria, Lecture Notes in Computer Science Vol. 1330, p209-221, Springer, October/November 1997.
38. Using Bidirectionality to Reduce Message Passing in a Distributed Filtering Algorithm Y. Hamadi, Fifth International Conference on Advanced Computing (**ADCOMP'97**), Madras, India, p53-58, December 1997.
39. Improving GSAT behavior by using FPGA Y. Hamadi, D. Merceron, Fifteenth International Joint Conference on Artificial Intelligence (**IJCAI'97**), Nagoya Japan, August 1997.
40. Distribution de GSAT Y. Hamadi, Actes des Journées Francophones d'Intelligence Artificielle Distribuée et de Système Multi-Agents (**JFIADSMA'96**), Port Camargue, France, Editions Hermès p189-199, Avril 1996.
41. GSAT Distribution Y. Hamadi, C. Bessière and J. Quinqueton, Proceedings of the Second International Conference on Multi-Agents Systems (**ICMAS'96**), Kyoto, Japon, AAAI press, December 1996.

Workshops and others

1. Optimizing Architectural and Structural Aspects of Buildings towards Higher Energy Efficiency, A. Fialho, Y. Hamadi, and M. Schoenauer, (**GECCO'11**), Workshop on GreenIT Evolutionary Computation, July 2011.
2. Lazy Decomposition for Distributed Decision Procedures, Y. Hamadi, J. Marques-Silva, and C. M. Wintersteiger, International Workshop on Parallel and Distributed Methods in verifiCation (**PDMC'11**).

3. Building Portfolios for the Protein Structure Prediction Problem, A. Arbelaez, Y. Hamadi, and M. Sebag, Workshop on Constraint Based Methods for Bioinformatics (**WCB'10**), July 2010, Edinburgh, UK.
4. A SAT-based approach to decipher Gene Regulatory Networks, F. Corblin, L. Bordeaux, Y. Hamadi, E. Fanchon, and L. Trilling, (**RIAMS'07**).
5. Solving Configuration Problems in Excel, L. Bordeaux and Y. Hamadi, (**AAAI'07**) Int. Workshop on Configuration.
6. Delegation in Tree-search for Distributed Constraint Satisfaction, M. Basharu, K. Brown, and Y. Hamadi, **IJCAI'07** Eight International Workshop on Distributed Constraint Reasoning (**DCR'07**).
7. Multi-Directional Distributed Search with Aggregation, G. Ringwelski and Y. Hamadi, IJCAI'05 Sixth International Workshop on Distributed Constraint Reasoning (**DCR'05**).
8. Distributed Constraint Satisfaction applied to Log-based Reconciliation, Y. Chong and Y. Hamadi, IJCAI'05 Sixth International Workshop on Distributed Constraint Reasoning (**DCR'05**).
9. Cycle-cut decomposition and log-based reconciliation, Y. Hamadi, in 14th International Conference on Automated Planning & Scheduling (**ICAPS'04**), Workshop: Connecting Planning Theory with Practice, Whistler, British Columbia, Canada, June 3-7 2004.
10. An Overview of the Gridline Project, Y. Hamadi, A. M. Frischand I. Miguel, in 14th International Conference on Automated Planning & Scheduling (**ICAPS'04**), Workshop: Planning and Scheduling for Web and Grid Services, Whistler, British Columbia, Canada , June 3-7 2004.
11. DisCSPs: Tools for efficient and generic multi-agents negotiations, 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (**WECWIS'02**), p245-248 Newport Beach, June 2002.
12. EOLE project: Toward an on-line optimization framework dedicated for Telecom domain EOLE consortium, Seventh International Conference on Principles and Practice of Constraint Programming (**CP01**), Workshop on On-Line combinatorial problem solving and Constraint Programming (OLCP'01), October 2001.
13. Traitement des problèmes de satisfaction de contraintes distribués, PhD thesis, LIRMM/CNRS, Montpellier, France.