Ten years of parallel theorem proving: a perspective

Maria Paola Bonacina [*]

Department of Computer Science – The University of Iowa
Iowa City, IA 52242-1419, USA
E-mail: bonacina@cs.uiowa.edu

1 Introduction

The field of Strategies for Automated Deduction is concerned with the *control* of deduction, that is, with the *search strategies* employed to search for a solution, e.g., a proof in theorem proving, a model in model building, a normal form in term rewriting. The definition of the search problem in theorem proving is well known. The availability of a sound and refutationally complete inference system I guarantees the existence of a proof (a derivation of a contradiction), for any inconsistent set $H \cup \{\neg\varphi\}$, where H is a set of assumptions and φ a conjectured theorem. However, given an initial state with H and $\neg\varphi$, I can generate many derivations, because an inference system is *non-deterministic*. The search problem is how to control I so that a proof can be found (*fairness*) using as little resources as possible (*efficiency*). A *search plan* Σ is the mechanism that decides at each step how to apply I, so that the derivation generated by I and Σ from an initial state is unique, and the combination of inference system and search plan forms a *deterministic* procedure called a *theorem-proving strategy*.

A taxonomy of fully-automated, general-purpose, first-order theorem-proving strategies *based on how they search* was presented in [3], and Figure 1 shows its graphical representation. We call *ordering-based strategies* those strategies that work on a set of objects (e.g., clauses) and develop implicitly many proof attempts, and *subgoal-reduction strategies* those strategies that work on one object at a time (e.g., a goal clause, a goal literal, a sequent) and develop one proof attempt at a time (e.g., a tableau), backtracking when the current proof attempt cannot be completed into a proof. Ordering-based strategies never backtrack, because whatever they do may further one of the proof attempts (see Table 1).

Within this taxonomy a formal definition of search plan was developed and instantiated for all classes of strategies. Let Θ be a first-order signature, $\mathcal{L}_\Theta$ a Θ-language, $\mathcal{P}(\mathcal{L}_\Theta)$ its powerset, and *States* the set of all possible states of a theorem-proving search problem $H \cup \{\neg\varphi\}$. Depending on the strategy states may be sets of clauses, or tuple of components. A search plan Σ is made of at least three components:

– A *rule-selecting function* $\zeta\colon States^* \to I$, which selects the next rule to be applied based on the history of the search so far;

[*] Supported in part by the National Science Foundation with grant CCR-97-01508.

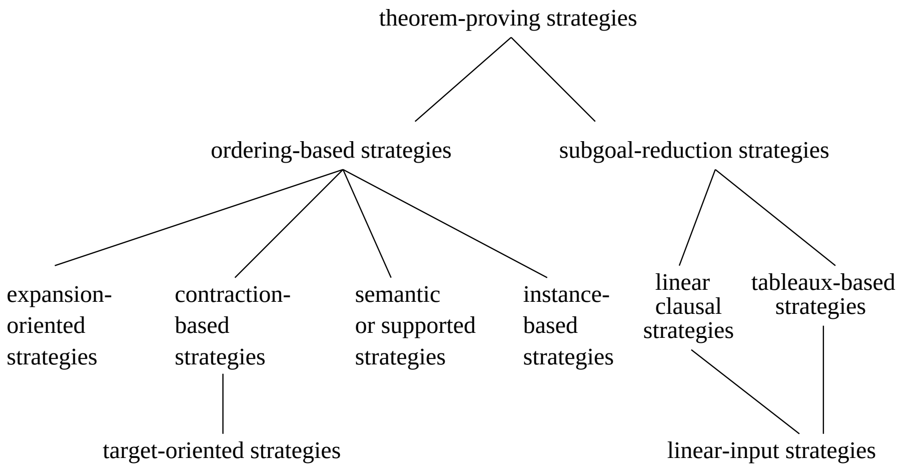
theorem-proving strategies
ordering-based strategies
subgoal-reduction strategies
expansion-oriented strategies
contraction-based strategies
semantic or supported strategies
instance-based strategies
linear clausal strategies
tableaux-based strategies
target-oriented strategies
linear-input strategies

Fig. 1. Classes of strategies

	Ordering-based	Subgoal-reduction
Data	set of objects	one goal-object at a time
Proof attempts built	many implicitly	one at a time
Backtracking	no	yes
Contraction	yes	no

Table 1. Two main classes of strategies

- A *premise-selecting function* $\xi\colon States^* \times I \rightarrow \mathcal{P}(\mathcal{L}_\Theta)$, which selects the elements of the current state the inference rule should be applied to;
- A *termination-detecting function* $\omega\colon States \rightarrow Bool$, which returns *true* if the given state is successful (e.g., empty clause generated, tableau closed), $false$ otherwise.

If the current state is not successful, ζ selects a rule f and ξ selects premises $\psi_1 \ldots \psi_n$, the next step will consist of applying f to $\psi_1 \ldots \psi_n$. The sequence of states thus generated forms the derivation by $\langle I, \Sigma\rangle$ from the given input.

Our objective is to extend the taxonomy of [3] to *parallel strategies*, and develop notions of *parallel search plans*. A basic motivation for this undertaking is that parallelism affects precisely the control of deduction; additional motivations come from *strategy analysis* and *engineering of theorem provers*. Strategy analysis is a new field whose goal is the machine-independent evaluation of theorem-proving strategies; approaches to strategy analysis involve modelling and analyzing the searches produced by the various types of strategies, and therefore hinge on a search-based classification. Extending these tools (e.g., classification, notion of search plan, model of search) to parallel strategies supports the extension of the analysis itself (e.g., [1]). The engineering of theorem provers stands to benefit from any development of formal tools to specify search plans, because theorem-proving methods are too often specified in terms of inference rules only, resulting in under-specification of the problem on one hand, and loss of knowledge on the other, since much information on search in theorem proving remains hidden in the implementations.

This is an extended abstract of [2], which we refer to for a survey of parallel strategies, comparison with related work, and a bibliography of over seventy references.

2 Principles of parallelization

Theorem proving can be parallelized in many ways: a distinction among *parallelism at the term level*, *parallelism at the clause level*, and *parallelism at the search level* was introduced in [4]. This classification is based on the granularity of the parallel operations and the data they manipulate, so that these three types of parallelism aim at representing fine-grain, medium-grain and coarse-grain parallelism for deduction, respectively (see Figure 2).

In parallelism at the term level, the data accessed in parallel are subexpressions of a formula such as terms or literals (here a "formula" means the basic unit of the language the inference system is for, e.g., a clause for clausal strategies, an equation for pure equational strategies), and the parallel operations are subtasks of an inference step. Thus, parallelism happens *below the inference level*, as in *parallel matching*, *parallel unification* and *parallel term rewriting*. The rationale for parallelism at the term level is that since a strategy executes these low-level operations very frequently, if one can make them very fast by parallelism, the overall performance of the strategy should improve.

In parallelism at the clause level, the data accessed in parallel are formulae, and the parallel operations are inferences, so that parallelism is *at the inference level*. Theorem-proving methods with *parallel inferences within a single search* (e.g., parallel resolution steps) belong to this category. The motivation is to speed-up the execution of the strategy by doing many inferences at each step.

Parallelism at the search level means *parallel searches*, that is, multiple deductive processes, each executing a strategy, developing a derivation and building its own set of data, search in parallel the space of the problem until one of them finds a proof. Approaches to parallel search are distinguishable based on how they differentiate and combine the activities of the deductive processes. One possibility is to subdivide the search space among the processes by subdividing the inferences or decomposing the problem: we use *distributed search* for this principle, using the word "distributed" in its literal meaning of "giving each a share of something." Distributed search aims at obtaining a speed-up over sequential search by ensuring that each parallel process has to search only a part of the whole space.

Another possibility is to let each process handle the problem in its entirety and differentiate them by having each process use a different search plan: we call this principle *multi-search*, because multiple plans are applied. The intuition be-

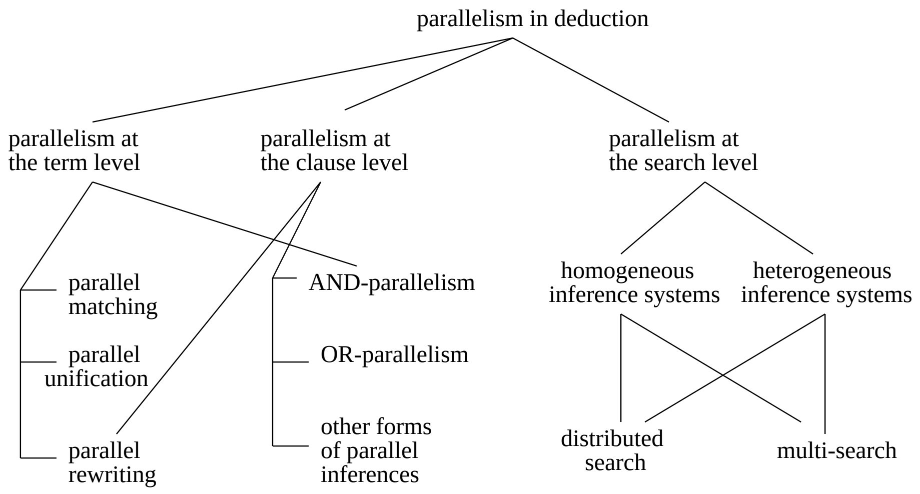
parallelism in deduction
parallelism at the term level
parallelism at the clause level
parallelism at the search level
parallel matching
parallel unification
parallel rewriting
AND-parallelism
OR-parallelism
other forms of parallel inferences
homogeneous inference systems
heterogeneous inference systems
distributed search
multi-search

Fig. 2. Types of parallelism in deduction

hind multi-search is that parallel processes executing different search plans will search the space of the problem in different order. Thus, multi-search aims at obtaining a speed-up over sequential search by letting each process take advantage of data earlier than its search plan would allow, because such data has been generated and communicated by other processes following different plans. Note that also distributed search may induce this effect, because a process barred from exploring a certain part of the search space may reach sooner deeper parts of its allowed portion, and send to other processes not only data that they would not generate because of the partition, but also data that they would generate much later. Distributed search and multi-search are not mutually exclusive: a strategy may feature instances of both principles.

A third option is to let each process have the same problem and search plan, but assign them different inference systems, leading to what is called a *heterogeneous* system. A motivation for heterogeneous systems is to use parallelism to combine subgoal-reduction and ordering-based inferences, typically by enabling subgoal-reduction processes to use clauses generated by the ordering-based processes as *lemmas.* If an heterogeneous system is combined with multi-search, each process executes a different theorem-proving strategy. Parallel search approaches where all processes have the same inference system are called *homogeneous.*

In Figure 2, *parallel term rewriting* overlaps with both parallelism at the term level (the data accessed in parallel are terms), and parallelism at the clause level (if each rewrite step is regarded as an inference). Thus, its classification is affected also by the application: in a context where the whole computation is a reduction one may consider parallel term rewriting as parallelism at the clause level; in the context of theorem proving, where the whole computation is better seen as a search, and each normalization, rather than each rewrite step, may constitute an inference, it is more natural to consider parallel term rewriting as parallelism at the term level.

The classification of *AND-parallelism* and *OR-parallelism* depends on the granularity: the traditional concepts of AND-parallelism (try in parallel the conjuncts of the current goal) and OR-parallelism (apply in parallel many rules to the selected literal of the current goal) are applied within one derivation, one search process, and therefore are instances of parallelism at the clause level. In fact, AND-parallelism may also be considered as parallelism at the term level, since the data accessed in parallel are the literals of a goal clause, hence subexpressions of formulae. However, envision a collection of parallel search processes, each developing its own derivation (e.g., its own sequence of tableaux) and having different AND-rules (i.e., different ξ_1) or different OR-rules (i.e., different ξ_2): then such situations represent forms of multi-search.

While in principle everything can be implemented in either shared memory or distributed memory, most methods with parallelism at the term or clause level use shared memory, and most parallel search methods use distributed memory. Tools for parallel programming, however, often let the high-level application programmer ignore whether the memory is physically shared or distributed, so that what is relevant is not the physical memory but the logical view of the memory: for instance, for ordering-based strategies, whether the method assumes a shared database of clauses, or separate databases and communication by message passing. In parallel search the processes develop separate derivations, and therefore require separate databases and communication by message passing; the latter can be implemented over a network or in a shared memory. Thus, *distributed deduction* and *distributed strategies* have also been used for methods with parallelism at the search level, especially those with distributed search.

3 A taxonomy of parallel theorem-proving strategies

While parallel theorem proving is still a young field, several approaches to parallelization have been tried for various classes of strategies: Figure 3 summarizes the taxonomy in [2], and combines Figures 1 and 2, with dotted lines linking types of parallelism and classes of strategies they have been applied to.

The heart of the study in [2] is to consider how the principles of parallelization of Section 2 affect the notion of search plan: given a sequential strategy $\mathcal{C} = \langle I, \Sigma \rangle$, assume that we denote by $\mathcal{C}' = \langle I, \Sigma' \rangle$ its parallelization; then the question is what is Σ' depending on the applied parallelization principle.

For parallelism at the term level, we find that exactly because the term level is *below* the level where the search plan makes decisions, the search plan is unaffected by the parallelization. Most fine-grained methods choose one of two approaches. The first one consists of *replacing* the search plan by a low-level data-driven form of concurrency, where all conflict-free inferences happen in parallel; an example is concurrent rewriting. The second approach is to be *strategy-compliant* in the sense that the parallel strategy is guaranteed to execute inferences in the same order as the sequential strategy, so that $\Sigma' = \Sigma$, and $\mathcal{C}' = \mathcal{C}$.

For parallelism at the clause level, a study of several parallel methods, both ordering-based and subgoal-reduction based, shows that the control of the parallel strategy is obtained by considering the actions of the sequential strategy as *tasks* (e.g., process a given-clause, solve a subgoal), and designing the parallel search plan as a *scheduler* that assigns tasks to parallel processes. Indeed, parallelism at the clause level parallelizes the inferences within one derivation, which are precisely what the search plan is supposed to order; thus, a parallel search plan for parallelism at the clause level is essentially a scheduler that assigns inferences to parallel processes.

In parallelism at the search level, each process generates a derivation, and therefore needs to execute a search plan. In addition to deduction, parallel search involves controlling *communication* (for both multi-search and distributed search) and *subdivision of the work* (for distributed search). Distributed search needs communication to preserve completeness and load-balance; multi-search needs it to allow every process to take advantage of the results of others, and also for completeness, if some processes employ unfair search plans.

In [2], we classify strategies based on whether the control of deduction and the control of parallelism are separate, or combined, because in the first case the search plan is responsible only for the control of deduction like in the sequential case. An approach to *separate the control of deduction and the control of parallelism* is to establish a *hierarchy* of parallel processes (e.g., master and slaves), with different processes playing different roles. The study in [2] covers master-slaves methods for distributed-search ordering-based strategies, multi--search ordering-based strategies, multi-search subgoal-reduction strategies, and heterogeneous systems. In most methods, each slave executes a sequential search plan to generate its derivation, and all other control issues (e.g., subdivision, communication, selection of "good" data, user interface) are dealt with in a centralized way by the master, which does not perform deductions. Thus, a parallel search plan for such strategies can be seen as a collection of sequential search plans, one per slave: $\Sigma' = \langle \Sigma_1, \ldots, \Sigma_n \rangle$, with $\Sigma_1 = \ldots = \Sigma_n$ in the case of distributed search with no multi-search.

Other parallel search methods have no hierarchy, and assume that all processes are *peers*. Among these, the survey in [2] covers distributed-search ordering-based strategies, multi-search ordering-based strategies, and heterogenous systems based on combinations of theorem provers. If all processes are peers, there is no central control, and the search plan executed by each process controls both deduction and parallelism. Thus, this is the situation that truly requires a *parallel search plan.*

Parallel search plans We begin by defining the notion of *search plan with communication*, which will form the common core to define on one hand a *multi-plan*, that is, a search plan for multi-search, and on the other hand a *distributed-search plan*, that is, a search plan for distributed search.

In order to describe unambiguously how a strategy generates a derivation, inference rules were characterized already in [3] as functions $f\colon \mathcal{P}(\mathcal{L}_\Theta) \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$, which take as argument a set of premises, say X, and return a pair of sets, the set of generated data to be added, $\pi_1(f(X))$, and the set of data to be deleted, $\pi_2(f(X))$, where $\mathcal{P}(\ )$ is powerset, and π_1 and π_2 are the projections $\pi_1(x,y) = x$ and $\pi_2(x,y) = y$. If f does not apply to X, $f(X) = (\emptyset, \emptyset)$.

A theorem proving strategy with parallel search features, in addition to an inference system I, a set M of *communication operators*, including at least *send* and *receive*. Since communication is among the responsibilities of the search plan, we intend to view communication acts as steps of the derivation. For this purpose, we need to define the communication operators in such a way that communication steps are homogeneous with inference steps. Thus, we define them as functions that take as argument the set of data being communicated, and return a set of data to be added and a set of data to be deleted, relative to the process executing the communication step:

– $\mathit{send}\colon \mathcal{P}(\mathcal{L}_\Theta) \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$ such that for all X, $\mathit{send}(X) = (\emptyset, \emptyset)$, because *send* does not modify the database of the sender; and
– $\mathit{receive}\colon \mathcal{P}(\mathcal{L}_\Theta) \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$, such that for all X, $\mathit{receive}(X) = (X, \emptyset)$, because *receive* adds the received data to the database of the receiver.

Alternative definitions are discussed in [2].

Since the search plan is in charge of communication, the rule-selecting function ζ may select not only inference rules, but also communication operators, yielding $\zeta\colon \mathit{States}^* \rightarrow I \cup M$. While this may be sufficient for some strategies, one may envision methods where rule selection keeps the identity of the process and the number of processes into account. Assuming that processes are denoted by natural numbers (e.g., if there are n processes, they are identified by $0, \ldots n-1$), we define $\zeta\colon \mathit{States}^* \times \mathbb{N} \times \mathbb{N} \rightarrow I \cup M$, where the second and third arguments are the identifier of the process which is doing the selection and the number of processes.

By applying similar considerations to the premise-selecting function ξ, we obtain that a *search plan with communication* for a set I of inference rules and a set M of communication operators has the form $\Sigma = \langle \zeta, \xi, \omega \rangle$, where

– $\zeta\colon \mathit{States}^* \times \mathbb{N} \times \mathbb{N} \rightarrow I \cup M$ selects an inference rule or a communication operator for the next step;
– $\xi\colon \mathit{States}^* \times \mathbb{N} \times \mathbb{N} \times (I \cup M) \rightarrow \mathcal{P}(\mathcal{L}_\Theta)$ selects a set of premises from the current state (e.g., $\xi((S_0, \ldots S_i), n, k, f) \subseteq S_i$); and
– $\omega\colon \mathit{States} \rightarrow \mathit{Bool}$ returns *true* if and only if the given state is successful.

If ζ selects *send*, the data chosen by ξ will be sent. If ζ selects *receive*, whichever data are pending to be received will be received, so that there is no need of selection of data for *receive*. If no data are pending, nothing is received (i.e., $\mathit{receive}(\emptyset) = (\emptyset, \emptyset)$). Thus, this notion of *receive* function models not only the act of receiving, but also the act of testing whether there are messages pending to be received, which is also a responsibility of a search plan with communication. This notion of search plan with communication applies to both ordering-based and subgoal-reduction strategies, once *States* is instantiated appropriately.

Since the essence of multi-search is to use multiple search plans and let each process take advantage of others' results through communication, a *multi-plan* is simply a collection of search plans with communication, one per process: $\Sigma = \langle \Sigma_0, \ldots, \Sigma_{n-1} \rangle$. The survey in [2] includes multi-plans where the ξ_i's employ different goal-oriented heuristics, or different heuristics to decide which clauses deserve to be broadcast, or select the given clause by different criteria.

A *distributed-search plan*, on the other hand, needs an additional component to handle the subdivision. Given a theorem-proving problem S, a subdivision of the search space among processes $p_0, \ldots p_{n-1}$ is a subdivision of the inferences in the *closure* $S^*_I = \bigcup_{k \geq 0} I^k(S)$ where $I^0(S) = S$, $I^k(S) = I(I^{k-1}(S))$ for $k \geq 1$, and $I(S) = S \cup \{\varphi | \ \varphi \in \pi_1(f(\varphi_1, \ldots \varphi_n)),\ f \in I,\ \varphi_1, \ldots \varphi_n \in S\}$. Since S^*_I is infinite and unknown, at each stage S_i of a derivation the search plan subdivides the inferences that can be done in S_i. Thus, the subdivision is built *dynamically* during the derivation. From the point of view of each process p_k, an inference is either *allowed* (it is assigned to p_k in the subdivision), or *forbidden* (it is assigned to others). Therefore, the subdivision of the search space can be modelled by distinguishing between allowed and forbidden steps, and Σ features a *subdivision function* α for this purpose. A minimal version of a subdivision function is $\alpha\colon \mathbb{N} \times I \times \mathcal{P}(\mathcal{L}_\Theta) \rightarrow Bool$, where $\alpha(k, f, X) = true/false$ means that p_k is allowed/forbidden to apply rule f to premises X. However, the above intuition that since S^*_I is not given, the subdivision is built dynamically, suggests that in general the subdivision function may keep the partial history of the derivation and the number of processes into account, yielding $\alpha\colon States^* \times \mathbb{N} \times \mathbb{N} \times I \times \mathcal{P}(\mathcal{L}_\Theta) \rightarrow Bool$.

Once dependency on the history of the derivation is introduced, it is more general to allow subdivision functions to be partial functions, according to the intuition that α may not be defined on premises that have not been generated. However, if ζ and ξ can select a certain f and X, α should be defined on their selection. For this purpose it would be sufficient to require that α is total on existing clauses, but we require that α is *total on generated clauses*, i.e., $\alpha((S_0, \ldots S_i), n, k, f, X) \neq \perp$ if $X \subseteq \bigcup_{j=0}^{i} S_j$, because it is undesirable that α "forgets" its decisions when clauses are deleted by contraction. Another requirement is that the subdivision function is *monotonic*, in the sense of not changing the status of a step after it has been decided:
$\alpha((S_0, \ldots S_i), n, k, f, X) \sqsubseteq \alpha((S_0, \ldots S_{i+1}), n, k, f, X)$, where $\sqsubset$ is the partial ordering $\perp \sqsubset false$ and $\perp \sqsubset true$.

Thus, a *distributed-search plan* for a set I of inference rules and a set M of communication operators has the form $\Sigma = \langle \zeta, \xi, \alpha, \omega \rangle$, where $\langle \zeta, \xi, \omega \rangle$ is a search plan with communication, and α is a subdivision function.

The next step is to relate sequential and parallel search plans: a search plan with communication $\Sigma' = \langle \zeta', \xi', \omega' \rangle$ *corresponds* to a sequential search plan $\Sigma = \langle \zeta, \xi, \omega \rangle$ for inference system I, if for all $(S_0, \ldots S_i) \in States^*$, n and k,

- if $\zeta'((S_0, \ldots S_i), n, k) \in I$, then $\zeta'((S_0, \ldots S_i), n, k) = \zeta(S_0, \ldots S_i)$ (whenever ζ' selects an inference rule, it selects the rule that ζ would select if given the same partial derivation),
- $\forall f \in I$, $\xi'((S_0, \ldots S_i), n, k, f) = \xi((S_0, \ldots S_i), f)$ (ξ' selects the premises that ξ would select if given the same partial derivation and inference rule), and
- $\omega' = \omega$.

A multi-plan $\Sigma' = \langle \Sigma'_0, \ldots, \Sigma'_{n-1} \rangle$ is a *parallelization by combination* of the sequential search plans $\Sigma_0, \ldots, \Sigma_{n-1}$ if for all i, $0 \leq i \leq n-1$, Σ'_i corresponds to Σ_i.

A distributed-search plan $\Sigma' = \langle \zeta', \xi', \alpha, \omega' \rangle$ is a *parallelization by subdivision* of a sequential search plan $\Sigma = \langle \zeta, \xi, \omega \rangle$ if $\langle \zeta', \xi', \omega' \rangle$ corresponds to Σ. The name "parallelization by subdivision" captures the fact that the derivations generated by the parallel processes differ among themselves, and from the sequential derivation that Σ would generate from the same input, because of the subdivision function, which for every process forbids some choices, forcing it to choose something else.

A *homogeneous parallel strategy* is defined by a triple $\langle I, M, \Sigma \rangle$, which yields a *multi-search strategy*, if Σ is a multi-plan $\langle \Sigma_0, \ldots, \Sigma_{n-1} \rangle$, and a *distributed-search strategy*, or *distributed strategy* tout court, if Σ is a distributed-search plan $\langle \zeta, \xi, \alpha, \omega \rangle$. A multi-search strategy $\langle I, M, \Sigma' \rangle$ is a *parallelization by combination* of n sequential strategies $\langle I, \Sigma_i \rangle$, for $0 \leq i \leq n-1$, if Σ' is a parallelization by combination of the Σ_i's. A distributed strategy $\langle I, M, \Sigma' \rangle$ is a *parallelization by subdivision* of a sequential strategy $\langle I, \Sigma \rangle$, if Σ' is a parallelization by subdivision of Σ. A strategy with both distributed search and multi-search can be modelled by defining a multi-plan which is a collection of distributed-search plans.

A *heterogeneous parallel strategy* is defined by a vector $\langle I_0, \ldots, I_{n-1}, M, \Sigma \rangle$, with an inference system per process, and the same options for the search plan. A *heterogeneous multi-search strategy* can also be written in the form $\langle \langle I_0, M, \Sigma_0 \rangle, \ldots, \langle I_{n-1}, M, \Sigma_{n-1} \rangle \rangle$, grouping together the inference system and the search plan for each process.

The following definition of parallel derivation covers distributed search, multi-search, their combination, and both homogeneous and heterogeneous inference systems. Given a theorem-proving problem S, the *parallel derivation* generated by strategy $\mathcal{C} = \langle I_0, \ldots, I_{n-1}, M, \Sigma \rangle$, with $\Sigma = \langle \Sigma_0, \ldots, \Sigma_{n-1} \rangle$, for processes $p_0, \ldots p_{n-1}$, is a collection of n *local derivations* $S = S^k_0 \vdash_{\mathcal{C}} S^k_1 \vdash_{\mathcal{C}} \ldots S^k_i \vdash_{\mathcal{C}} \ldots$ for $k \in [0, n-1]$, such that for all k and $i \geq 0$, if

- $\omega^k(S^k_i) = false$,
- $\zeta^k((S^k_0, \ldots S^k_i), n, k) = f \in I_k$,
- either $f = receive$ and there is a set of formulae X pending to be received, or $f \neq receive$ and $\xi^k((S^k_0, \ldots S^k_i), n, k, f) = X$, and
- either $f \in M$ or $\alpha^k((S^k_0, \ldots S^k_i), n, k, f, X) = true$,

then $S^k_{i+1} = S^k_i \cup \pi_1(f(X)) - \pi_2(f(X))$.

The strategy, and the derivation, is homogeneous, if $I_0 = \ldots = I_{n-1}$, heterogeneous, otherwise. When $\omega^k(S^k_i) = true$, process p_k sends a halting message to all other processes (note that for a homogeneous inference system it is

$\omega^0 = \ldots = \omega^{n-1}$, because the test for the generation of a refutation is the same for all processes). For simplicity, the definition has been given for states made of a single component, but one can replace the S_i's with suitable tuples for the different types of strategies, either ordering-based or subgoal-reduction.

If $\Sigma = \Sigma_0 = \ldots = \Sigma_{n-1} = \langle \zeta, \xi, \alpha, \omega \rangle$, $\mathcal{C}$ is a distributed-search strategy, and the above definition yields a *distributed-search derivation* or *distributed derivation* for short.

If the Σ_i's are not all equal, and for all i, $0 \leq i \leq n-1$, α_i is the constant function equal to *true* on all arguments (hence no subdivision), $\mathcal{C}$ is a multi-search strategy, and the above definition yields a *multi-search derivation.*

The above definition of parallel derivation captures the fact that all processes are *peers*, because every process p_k makes its decisions *locally*; indeed, the functions ζ^k, ξ^k, α^k, and ω^k apply to the partial history of the local derivation generated by p_k.

4 Discussion

Our previous analysis [4] considered factors such as size and monotonicity of the database during a derivation, conflicts among concurrent inferences, and the presence of *backward-contraction*, to suggest that parallelism at the search level may be overall the most promising, at least for ordering-based strategies.

In this paper, we have extended our analysis to the impact of the parallelization approaches on the control of search. We observed that approaches with parallelism at the term level may replace the search plan by low-level data-driven forms of concurrency, or produce strategy-compliant parallelizations. It seems that the potential problem is a loss of control for the former, and an excess of control for the latter. Data-driven concurrency may be appropriate for ground computations that are guaranteed to converge (e.g., computing a congruence closure for ground completion), but may represent a counter-productive loss of control in general theorem proving, where the essence, from a practical point of view, is not saturation (i.e., do all the steps, with the order being a secondary issue), but effective search (i.e., find a good order to do the steps in order to avoid doing them all). Strategy-compliant parallelizations, on the other hand, may be too conservative: they avoid the risk of mixing search with parallelism, but they renounce using parallelism to try to generate better searches.

Approaches with parallelism at the clause level turn the search plan into a *scheduler* of parallel inferences, assigning inferences – viewed as *tasks* – to a pool of parallel processes. This type of approach may have been appealing because it allowed to apply to theorem proving general scheduling techniques (e.g., *task stealing*). However, such techniques may not take specific theorem-proving knowledge into account (e.g., the differences between expansion tasks and contraction tasks). More importantly, the potential problem is one of granularity, that is, whether processing a given clause or a subgoal is a sufficiently large task. Intuitively, the risk is that such tasks are too small with respect to the amount of work required by the theorem-proving problem, so that too much time is spent in task scheduling and not in inference making. Furthermore, this problem may become worse as the difficulty of the theorem-proving problem grows (e.g., requiring to process tens of thousands of given clauses or subgoals), against the expectation that parallel theorem proving makes a difference precisely on the hardest searches.

In summary, the analysis of parallelization principles from the point of view of the search plan component of theorem proving, seems to confirm the philosophy of [4] in favor of parallelism at the search level. The evolution of the field since 1992, when the material in [4] was first written, appears to concur with this view: the survey in [2] shows a movement from fine or medium-grain parallelism towards coarse-grain parallelism, and from master-slaves hierarchies to peer processes.

Of course, parallel search is not free of obstacles. Problems include the *cost of communication*, *overlapping searches* and *scalability*. For the first one, a process may reach a proof faster exactly because it can ignore data handled by others; on the other hand, a delay in receiving an important clause (e.g., a good simplifier) may cause a process to perform redundant inferences. For the second one, if the parallel processes end up exploring overlapping areas of the search space, their efforts are partly wasted. In distributed search, the strategy induces a subdivision of the infinite unknown search space lying ahead by subdividing the generated search space. In these conditions, it is impossible to partition the search space into disjoint parts, and it is also difficult to minimize the overlap, especially in the search spaces of theorem proving, which are very redundant (i.e., there are many different ways of generating the same logical consequence). In multi-search, a source of overlap is that the search plans may not be sufficiently different: seemingly different search plans may turn out to generate searches with large overlap on some inputs. Part of the problem is that most known fair search plans are exhaustive, and it may be rare to get significantly different searches from plans that are all exhaustive in nature. The scarcity of truly diverse search plans may also be a reason why papers on multi-search methods rarely report experimental results for more than two or three processes. For parallel search to be beneficial, however, it may not be necessary to limit the overlap globally: it may be sufficient that the searches are different early on, so that a process may get sooner some important clause from some other process.

Scalability is difficult in parallel search, because the addition of a new process may change dramatically the searches of the others. In distributed search, one would expect that if we add more processes the performance improves, because each process should get a smaller portion of space to search. However, this is not guaranteed to happen, because increasing the number of processes changes the subdivision not only quantitatively, but also qualitatively. That is, the search space allowed to process p_k in a search with $2n$ processes may be radically different than the search space allowed to p_k in a search with n processes. The performance is not guaranteed to improve, because the subdivision with $2n$ processes may be worse from the point of view of finding a proof (e.g., it may break the search space in a way that prevents p_k from finding the proof found with

n processes). In multi-search, enriching the pool with an additional search plan may not help, because the added search plan may not be good for the problem, or may not interact usefully with the other plans. Similar considerations apply to heterogeneous systems. One may regard the lack of scalability as a price to pay for the possibility of super-linear speed-up by parallel search. Super-linear speed-up is possible because the parallel strategy has a different search plan than its sequential counterpart.

Parallel theorem proving may be now ten years old (1989–1999), which is a very short time for a field of research. Therefore, there are many directions for further research. For instance, Figure 3 shows that not much work has been done in combining parallel search with semantic strategies, where one may envision parallel processes reasoning under different interpretations. Also, while multi-search has been applied to both ordering-based and subgoal-reduction strategies, distributed search does not seem to have been applied to the latter. A possible explanation comes from the differences in Table 1: since ordering-based strategies work with a set of objects, and build many proof attempts implicitly, it is quite natural to think of subdividing the set of objects, or, better, the inferences they permit, in order to subdivide the search space, hence the proof attempts. Subgoal-reduction strategies work on one goal object at a time, building a proof attempt at a time, so that the idea of a subdivision may be less natural, or may lead to fall back on OR-parallelism or other forms of parallelism at the inference level, if the subdivision is done within the single proof attempt. The application of distributed search to subgoal-reduction strategies will require to design distributed-search plans for such strategies. In addition to new directions in design, there is the entire area of analysis of parallel strategies. A formal analysis of subdivision, overlap and communication was begun in [1], applying the bounded search spaces methodology to distributed-search contraction-based strategies. We hope that the formalization of parallel search plans in this paper will be a preliminary step towards more analyses of parallel-search strategies.

Acknowledgements The author thanks the Fakultät Informatik of the Technische Universität Dresden where this work was completed.

References

1. Maria Paola Bonacina. Analysis of distributed-search contraction-based strategies. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *Proc. of the 6th JELIA*, volume 1489 of *LNAI*, pages 107–121. Springer, 1998. Full version available as Tech. Rep. 98-02, Dept. of Comp. Sci., Univ. of Iowa, April 1998.
2. Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. Technical report, Dept. of Comp. Sci., Univ. of Iowa, May 1999.
3. Maria Paola Bonacina. A taxonomy of theorem-proving strategies. In Manuela Veloso and Michael J. Wooldridge, editors, *Artificial Intelligence Today*, volume 1600 of *LNAI*. Springer, 1999. To appear.
4. Maria Paola Bonacina and Jieh Hsiang. Parallelization of deduction strategies: an analytical study. *Journal of Automated Reasoning*, 13:1–33, 1994.

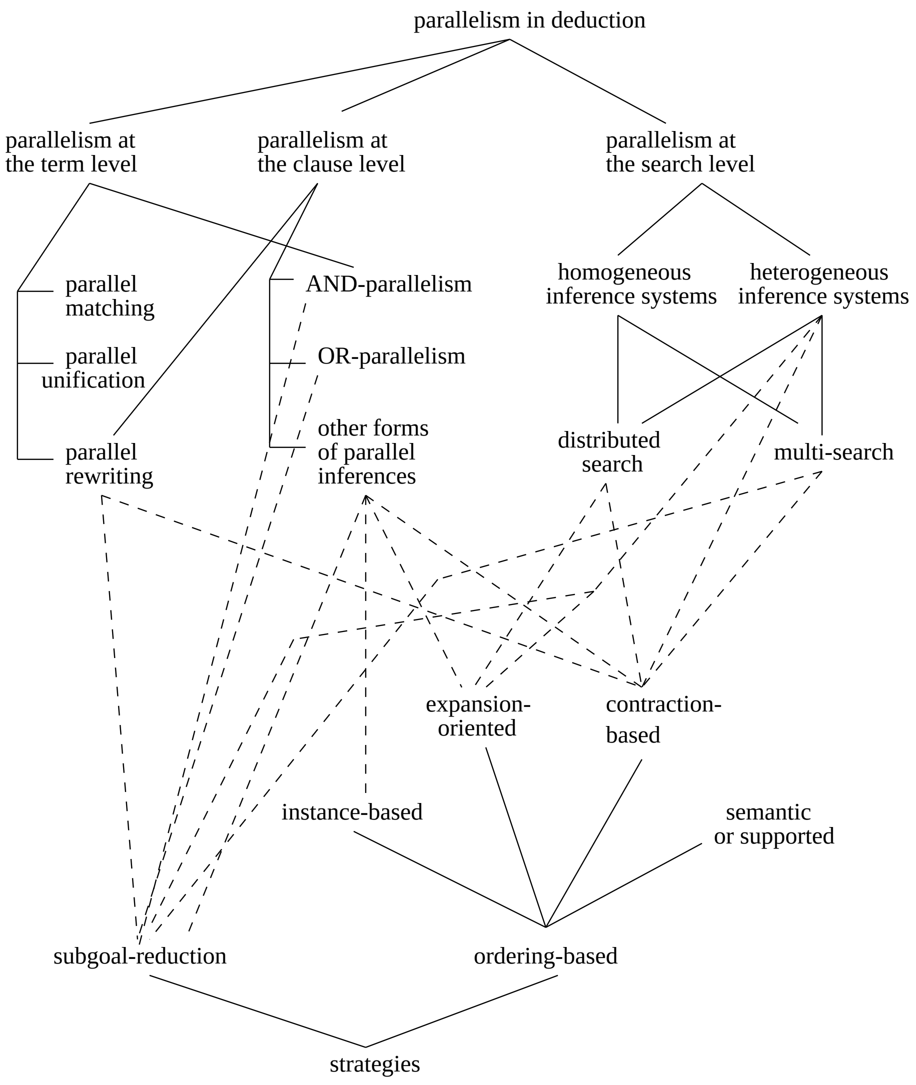

Fig. 3. Matching parallelization principles and classes of strategies