TRT - INRIA - ONERA

ANR-09-COSI-002 Technical Report

DESCARWIN

The Marriage of Descartes and Darwin

D5.4: Inoculation

THALES

Inria INVENTEURS DU MONDE NUMÉRIQUE

ONERA THE FRENCH AEROSPACE LAB

DOCUMENT CONTROL

	–: May 31, 2013	A:	B:	C:
Written by Signature	Yann Semet Pierre Savéant			
Approved by Signature	P. Savéant			

Revision index	Modifications
–	initial version
A	
B	
C	

1. PURPOSE

This document provides the description and experimentation of work conducted to improve the initialization phase of DaE's evolutionary engine. We focus, in particular, on initialization procedures built around an individual provided either by specific construction or by invoking external planners. Following the biological metaphor inherent to evolutionary algorithms, such procedures are usually called "artificial insemination" or "inoculation" in the literature, we choose to stick to the latter. Two experimental campaigns are reported here.

The first one is our early attempt at designing an inoculation procedure that builds the initial population around an individual constructed by our embedded planner, YAHSP, with an additional routine designed to produce a consequently refined estimate of the critical parameter defining YAHSP's exploratory allowance once the search has started, b_{max}. Experimental results show very poor performance and high risk of runs, with "unlucky" random seeds, getting stuck in fruitless sampling of the search space.

The second set of experiments follows a different strategy, getting rid of YAHSPas the inoculant provider on one hand and being far more parsimonious with b_{max} on the other hand. We use additional sampling both to compensate for b_{max}'s restricted increase and to produce the inoculant. We call this procedure "Inoculation through mass selection". Experimental results exhibit both substantial improvements in speed and risk reduction for our algorithm, albeit on a limited subset of the entire IPC benchmark.

CONTENTS

1 Purpose **3**

2 Introduction **5**
2.1 Context 5
2.2 State of the Art: The Seven Motivations of Highly Effective Inoculations 5

3 Inoculation with Yahsp **8**
3.1 Formalizing inoculation procedures 8
3.2 Meet Adam! 8
3.3 Experiments 10

4 Inoculation through mass selection **15**
4.1 Motivation and principle 15
4.2 Experiments 15
4.2.1 Consequences of increasing bmax 16
4.2.2 Failure rate 16
4.2.3 Start overhead of successful runs 16
4.2.4 Average speed over all runs 18

5 Conclusion **20**

6 References **21**

2. INTRODUCTION

"It is difficult to catch a black cat in a dark room, especially when it is not there."

Chinese Proverb

2.1 CONTEXT

Search is about finding things that are difficult to locate. Stochastic search, evolutionary or otherwise, is about injecting randomness in the process when the field is so large or labyrinthine one has no other choice anyway. The challenge however, like the devil, is in the details of that randomness. There are many things one can do to improve one's odds of finding what one is looking for: throwing more light in, work in teams, use binoculars, ask someone, run faster, search deeper and so forth. All of these tricks have counterparts in mathematical optimization and computational search: local sampling, parallel search, depth first search, external expert knowledge, relaxed constraints, hybrid algorithms, etc. It all boils down to injecting whatever form of intelligence, knowledge or power is available at reasonable cost in the stochastic process to tame and steer randomness in the right direction.

Evolutionary algorithms offer many such avenues of improvement. They are usually made of many interconnected heuristically designed modules, each of which performing some portion of the traditional genetic loop with some flavor of randomness: initialization, selection, reproduction, blind variations, replacement, stopping criteria. Representation, helper or hybridized algorithms and genotype mappings can also benefit from efforts to build an intelligent bias for random search. *DaE* makes no exception: it is an evolutionary algorithm coupled with an embedded AI planner. It has a convoluted workflow, five variation operators and dozens of parameters and routines, many of which have been designed or tuned with somewhat arbitrary randomness based ingredients which can be improved with externally provided intelligence.

In that perspective, the traditional evolutionary building block the present report deals with is *initialization*. More specifically, we look at how the initial population, and therefore the entire algorithm, can be improved by being made less pointlessly random from the start. The idea is to find a way to start the search directly from an interesting zone instead of blindly sampling the search space as is usually done by default. One possible way to do this is known as *inoculation*. In biology or medicine, inoculation refers to "the placement of something that will grow or reproduce, and is most commonly used in respect of the introduction of a serum, vaccine, or antigenic substance into the body of a human or animal, especially to produce or boost immunity to a specific disease". In our mathematical context we will seek to find a way to quickly construct one individual with a good fitness value or good building blocks and use variations of that individual to constitute the initial population. That individual, which we will call the *inoculant* will serve as the vehicle for the injection of interesting search space information into the genetic population where it will be spread and sampled along with successive generations. Obviously enough, this process relies on the assumption that we can design a way to fabricate this individual both quickly enough and with enough interesting material for inoculation to be worth the extra cost in algorithmic terms.

To sum up, and come back to our metaphoric epigraph, we will henceforth try to catch the cat faster by starting at least from the right room.

2.2 STATE OF THE ART: THE SEVEN MOTIVATIONS OF HIGHLY EFFECTIVE INOCULATIONS

As mentioned earlier, the main rationale behind inoculation is to steer randomness in the right direction by injecting external knowledge into your population. There are several such forms of knowledge and just as many potential benefits to motivate you to try to improve your algorithm with an inoculation procedure:

Motivation 1 : Bypassing "Senseless Randomness" Optimization fitness plots are usually very steep over early iterations or generations. It usually means that vast portions of the search space can be quickly discarded as being senseless for the problem at hand where interesting things only happen within the bounds of a restricted set of parameter intervals. A reasonable approximation of these intervals can usually be obtained quickly, either by design, heuristics or sampling, and inoculation, based on that estimate can serve the purpose of "starting directly from the bottom of the curve" which can represent a usually small but sometimes substantial fraction of the total run time. Another example where large savings can be obtained by cutting through trivial search has to do with order based approaches, such as permutation based algorithms typical of combinatorial optimization, where early generations are usually spent running for a first reasonable permutation which makes sense with respect to problem data. The *h1* heuristic used by DAEis an example of substantial improvement obtained in this manner: it prevents the consideration of decompositions which do not make sense in terms of earliest possible start dates for atoms.

Motivation 2 : Taking Time to Precompute An optimization problem is usually made of both generic domain information and specific settings for the problem at hand. Both contain many items that can be useful when solving the actual problem, i.e. when the search has started. These items can be secured through precomputations and later fed to the search algorithm, for instance by inoculation in the initial population. These precomputations can indeed often be modeled as partial solutions and be used to construct individuals. An example in AI planning is *landmarks*, which are facts that "must be true at some point on any solution path to the given planning task." [3]. Knowing such facts before starting the search can help restrict it to a smaller number of reasonable paths.

Motivation 3 : Solving a Relaxed Version of the Problem One fundamental technique in Operations Research and Mathematical Programming is to consider a relaxed version of the problem one is about to solve. Relaxed means without taking some or all constraints into account. Solutions to the relaxed problem and solutions to the actual problem often share many characteristics and the relaxed version, which can usually be obtained way more easily and quickly can be an excellent informant. It can, for instance, serve as the basis for an admissible heuristic providing a lower bound on the best possible solution in terms of the objective function, which can be used to guide the *A** algorithm in its search of a solution path. An inoculant based on a relaxed solution could be an excellent way to quickly identify elementary characteristics common to all good solutions.

Motivation 4 : Having a First Shot at the Actual Problem In a spirit similar to the previous motivation concerning relaxed solutions, if you have access to an external solver that is able to very quickly find a viable solution to your full problem, albeit with poor fitness performance, or if you can tune your solver to do that, you may have an excellent way to jumpstart your search! The creation of the inoculant in this case is moreover straightforward because what you obtain is already a complete, viable solution or individual.

Motivation 5 : Algorithm portfolios Extending over motivation 4, if you have several solvers with varying performance characteristics, the best possible strategy might be to combine their respective strengths with a hybrid scheme, possibly based on conditions identifying the best one use in a given context. Inoculation can act as one such strategy in several ways, as it can: 1) hybridize a fast but suboptimal solver with a deep solving one by giving a small initial time slot to the former and leaving the rest to the latter 2) hybridize metric sensitive solvers or solvers with abilities pertaining to different aspects of the solutions with similar time allowances in a relay race inspired fashion 3) allow many different solvers to offer a variety of inoculated starting points to be genetically mixed.

Motivation 6 : Solution Portfolios and Domain Experts Real-World optimization efforts could not be successful without domain experts. Algorithms, sophisticated as they might be, can only be efficient if they manipulate the right data in the right way and applying out-of-the-box algorithms on raw field data is bound to lead nowhere in the vast majority of cases. Any optimization campaign should be based on a dialogue between the computer scientist and the domain expert, the former incorporating information provided by the latter into the various parts of her algorithms and data structures. One thing experts often provide is solution patterns or portfolios: "in context XYZ, ABC is a good plan". These patterns often can

be seen as partial solutions and can be inoculated if one can come up with a way of constructing full individuals with these building blocks. An example of “primordial soup” of good building blocks serving as the basis for the initial population can be seen in Messy Genetic Algorithms [4].

Motivation 7 : Statistics and Sampling One thing one might want to attempt before launching the actual search is to use purely random sampling to get a quick idea of the shape of the search space with respect to the degrees of freedom at hand, an idea generalized by Estimation of Distribution Algorithms [1, 5] . There is always a compromise to be found in stochastic search between keeping on sampling and moving on to other parts of the search space and this compromise is usually particularly sensitive at the beginning of the search, especially with fast or prematurely converging algorithms. You may therefore want, in fear of local optima, to inoculate your initial population with strong individuals identified through fast but extensive preliminary surveys.

In spite of these numerous possible motivations, there are at least two potential drawbacks to inoculation that the designer will want to keep in mind. The first one takes place within the traditional quest evolutionary computation practitioners are all familiar with: the infamous exploration/exploitation tradeoff. If helping the search by having it start from an interesting zone can obviously speed things up, it can also turn out be an insidious bias leading to a local optimum and therefore be a source of premature convergence. Several countermeasures can be imagined: use several inoculants coming from different external sources, build an initial population filling routine with a carefully crafted use of variations to ensure sufficient variety, having a mixed scheme with both inoculants and purely random individuals and so on. The second drawback is that inoculation procedures are, most of the time, highly problem specific and therefore force you to make a step away from the admittedly utopic dream of a generic solver.

Extensive discussion of inoculation to boost evolutionary search can be found in [8, 7]. An example of inoculation in a real-world context with substantial benefits can be found in [6] where the technique is used to solve a railroad based combinatorial problem faster by using aforementioned motivations 1, 2 and 3.

3. INOCULATION WITH YAHSP

In this section, we describe and motivate the actual basic inoculation procedure we designed in the first place and present experimental results that show its inefficiency in several respects. We then introduce, also along with experiments, another way to boost our evolutionary initialization phase which brings improvements with respect to both stochastic risk and computation speed. We call this procedure Inoculation through Mass Selection (IMS).

3.1 FORMALIZING INOCULATION PROCEDURES

To make things more explicit, an inoculation procedure is made of the following components:

A substance provider This module is responsible for providing the substance to be injected in the population. It is usually a separate, externally provided module that is able to produce an individual or part of it.

An inoculant builder In case the provided substance is not a complete individual, a procedure is necessary to construct an full individual around it and insert it in the population. This individual is called the inoculant.

A variation scheme To ensure sufficient diversity for the genetic search to be exploratory enough, variations must be applied to the inoculant to produce variants in order to fill the population.

A composition scheme The initial population will be filled with inoculant variants but to an extent that can be defined by the algorithm designer. More elaborated schemes can also be imagined where several substance providers are put to work. The composition scheme will detail how the initial population will be composed between purely random individuals and the substance enriched inoculants provided by various sources.

3.2 MEET ADAM!

The initialization/inoculation procedure works as follows and as outlined by algorithms 1 through 4. Algorithm 1 gives the global outline of our initialization, which work in two separate phases: the initial population is first built by gathering randomly generated individuals and then, an estimation for b_{max}, a critical parameter setting the embedded planner's exploratory allowance, is computed by manipulating the population. As outlined in algorithm 2 described in more depth elsewhere [2], the random generation procedure follows an intermediate goal setting heuristic framed by a minimal timestamp computing heuristic, $h1$, and randomly fills the individuals' predefined goals with random atoms picked while respecting mutex constraints. In the normal case, without inoculation, algorithm 3 proceeds to estimate the proper value for b_{max} in an incremental manner: its value is increased until a sufficient number of feasible individuals is present int the population as an indivdual, i.e. a decomposition of the planning problem in intermediate goals, can be declared feasible or not depending on how extensively Yahsp is allowed to work on it, which is determined by b_{max}.

The inoculation procedure is an alternative to the aforementioned incremental b_{max} estimation procedure. It is detailed in algorithm 4. It works in several steps. First, it calls Yahshp for a quick solution on the entire planning problem. Then, this solution plan is converted to a decomposition with intermediate goals to make up an individual ready for genetic mixing. This new individual will serve as the *inoculant*, we call it *Adam* for reference purposes. The initial population is then reconstructed using identical copies of *Adam*. A double loop routine follows, both to introduce diversity and to produce a relevant estimate for b_{max}. Following a classical constraint handling technique, we try to push the population towards the infeasibility border. To do so, *delgoal* mutations are first applied to all individuals iteratively until a sufficient number of them are pushed to infeasibility. Then b_{max} is iteratively increased until the population reaches the specified ratio of feasible individuals back.

Algorithm 1 Initialization

Require: $N, b_{max}fixed, b_{max}ratio, b_{max}lastweight, b_{max}increase, insemination$ // initialization parameters
for $i \in [1..N]$ **do**
    $pop[i] \leftarrow GenerateIndividual(K)$ // random individuals with K intermediate goals
if $(b_{max}fixed \neq 0)$ **then**
    $b_{max} = b_{max}fixed$
else
    if $(insemination == 1)$ **then**
        $b_{max} = estimateb_{max}insemination(b_{max}ratio, b_{max}increase)$
    else
        $b_{max} = estimateb_{max}incremental(b_{max}ratio, b_{max}increase)$
$b_{max}last = b_{max} * b_{max}lastweight$

Algorithm 2 generateIndividual(N)

Require: T // candidate start times
$D \leftarrow \{\}$ // ordered list of timestamps
repeat
    $t \leftarrow \mathbb{U}(T)$
    $T \leftarrow T \setminus \{t\}$
    Insert(t, D) // maintain D ordered
until $\#D = N$
Ind $\leftarrow \{\}$ // start building the individual
for $t \in D$ **do**
    $s \leftarrow \{\}$ // start building the intermediate goal
    $A_t \leftarrow \{a \in A \mid T(a) = t\}$ // atoms that can appear at t
    $n \leftarrow \mathbb{U}([1, \#A_t])$ // number of atoms
    while $n \neq 0 \wedge A_t \neq \{\}$ **do**
        $a \leftarrow \mathbb{U}(A_t)$ // choose uniformly an atom in A_t
        $s \leftarrow s \cup \{a\}$ // add to s
        $A_t \leftarrow A_t \setminus (\{a\} \cup M(a))$ // remove all mutex
        $n \leftarrow n - 1$
    Ind $\leftarrow$ Ind $+ \{s\}$ // add the new intermediate goal
return Ind

Having done all that, the population is made of borderline feasible variants of Adam, a supposedly stronger than random individual, and b_{max} is set to a proper value. The search can start...

So in terms of our formalism:

Basic Inoculation	
Substance provider	YAHSP
Inoculant builder	1 to 1 decomposition
Variation scheme	*delgoal* mutations to infeasibilty
Composition scheme	All inoculant variants

Algorithm 3 $estimateb_{max}incremental$

Require: $N, b_{max}fixed, b_{max}ratio, b_{max}lastweight, b_{max}increase, b_{max}max, insemination$
1: $b_{max} \leftarrow 1$
2: $goodguys \leftarrow 0$
3: $goodguysMin \leftarrow N * b_{max}ratio$
4: **while** $((goodguys < goodguysMin)AND(b_{max} < b_{max}max))$ **do**
5: **for** $i \in [1..N]$ **do**
6: **if** $pop[i].isFeasible()$ **then**
7: $goodguys++$
8: $b_{max} *= b_{max}increase$
9: **return** b_{max}

Algorithm 4 $estimateb_{max}insemination$

Require: $N, b_{max}fixed, b_{max}ratio, b_{max}lastweight, b_{max}increase, b_{max}max, insemination$
1: $solutionPlan \leftarrow yahspCreateAdam()$
2: $Adam \leftarrow convertToDecomposition(solutionPlan)$
3: $eval(Adam)$
4: **for** $i \in [1..N]$ **do**
5: $pop[i] \leftarrow Adam$
6: $goodguys \leftarrow 0$
7: $goodguysMin \leftarrow N * b_{max}ratio$
8: **while** $(goodguys > goodguysMin)$ **do**
9: $goodguys \leftarrow 0$
10: **for** $i \in [1..N]$ **do**
11: $mutate(pop[i])$
12: $eval(pop[i])$
13: **if** $pop[i].isFeasible()$ **then**
14: $goodguys++$
15: **while** $((goodguys < goodguysMin)AND(b_{max} <= b_{max}max))$ **do**
16: $eval(pop)$
17: $b_{max} *= b_{max}increase$
18: **return** b_{max}

3.3 EXPERIMENTS

To evaluate the performance of our inoculation procedure, we conducted a number of experiments over several instances of several domains of the 7th International Planning Competition (IPC7). The bottom line of these experiments is that as such, our inoculation procedure performs very poorly compared to the normal initialization procedure. We report here a few significant experiments.

In all cases, we use *Divide-and-Evolve*'s default parameters : a population size of 100, a deterministic tournament of size 5, 700 offsprings, a goal neighborhood radius of 2, changeAtom and delAtom mutations both with 0.8 probabilities (per atom), a crossover probability of 0.2, mutation probability of 0.8 and weights for delgoal, addgoal, delatom, addatom of 1,3,1 and 1 respectively. We use a bmax ratio of feasibility of 1%, a bmaxlastweight of 3 and a bmaxincreasecoeff of 2.

For the first set of experiments, we launch *Divide-and-Evolve*with and without basic inoculation and center our observations on two parameters: the average fitness value for Adam as oppposed to the average fitness value of the best individual in a random population on one hand and the average speed at which the individuals can be processed, which we measure as the average time it takes to reach the first generation. All results are averaged over several seeds.

As can be seen in figures 3.1, results are consistent over all test domains, here Elevators, Openstacks and Floortile and over all instances, of increasing difficulty: Adam has a very poor fitness value, as exemplified by the Elevators P01 case where it starts around 4000 where the average random population starts around 2000. Fitness values are less contrasted with Openstacks but still give the same picture as do the Floortiles results. In terms of speed, the qualitative conclusion is the same and even more spectacularly so, except for

Floortile: inoculating Adam results in a considerably slower, up to a factor of 10, algorithm than proceeding with randomly generated individuals. Preliminary analysis suggest that Adam's poor performance is due to two factors: an excessive bmax due to the way its value is computed and an excessive length, probably due to a too straightforward inoculant builder.

Additional experiments, not reported here, show that in addition, runs of *Divide-and-Evolve*launched with inoculation are unable to improve fitness values efficiently over time: Adam is consistently bad regardless of how strongly one tries to perturb it and restrains the search very poor local minimum.

To sum up, our basic inoculation procedure offers three avenues of improvement: Adam is stupid, slow and stubborn. The next section offers an algorithmic variant that will try to address some of these shortcomings.

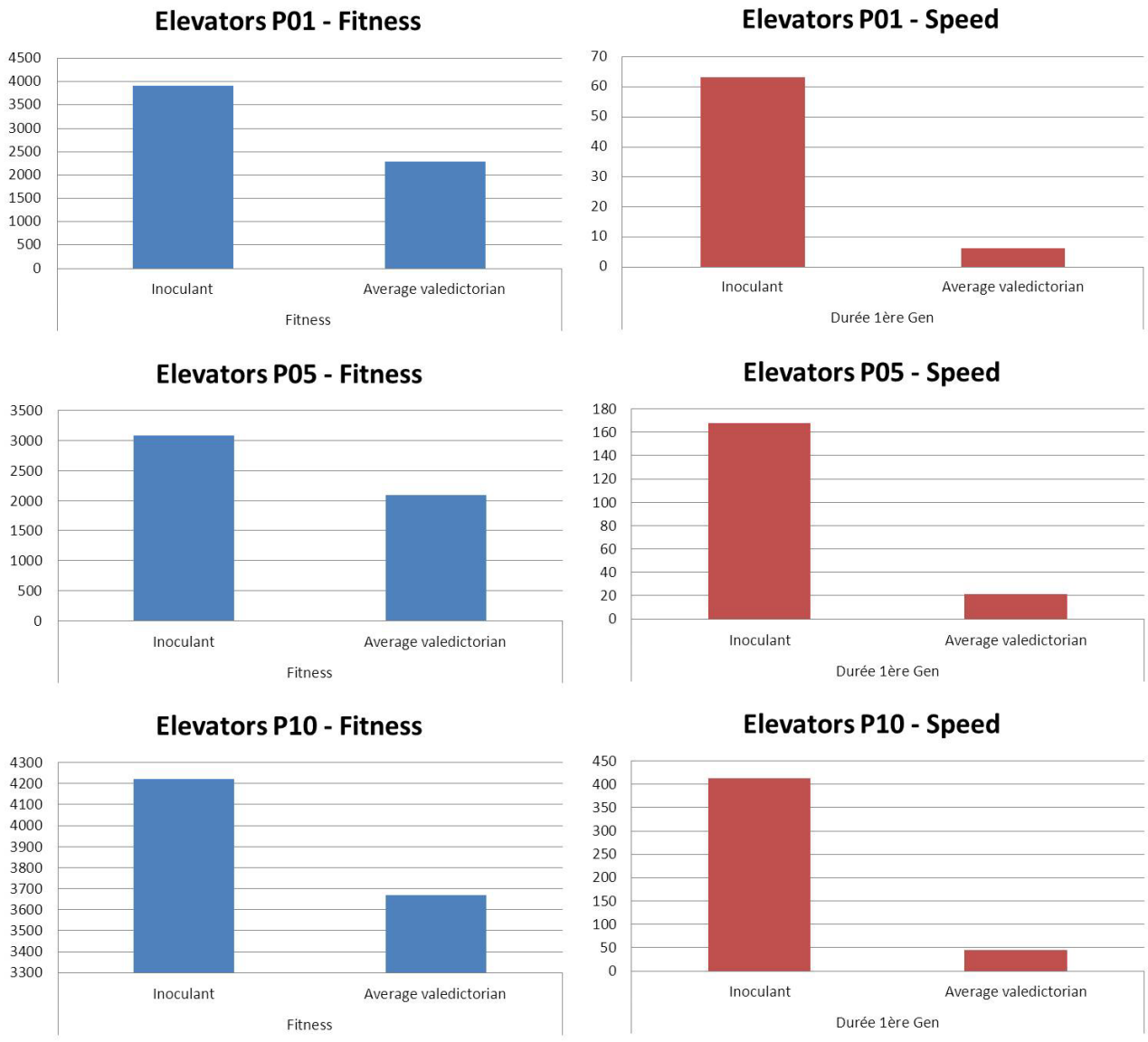

Figure 3.1: Basic inoculation performance on Elevators

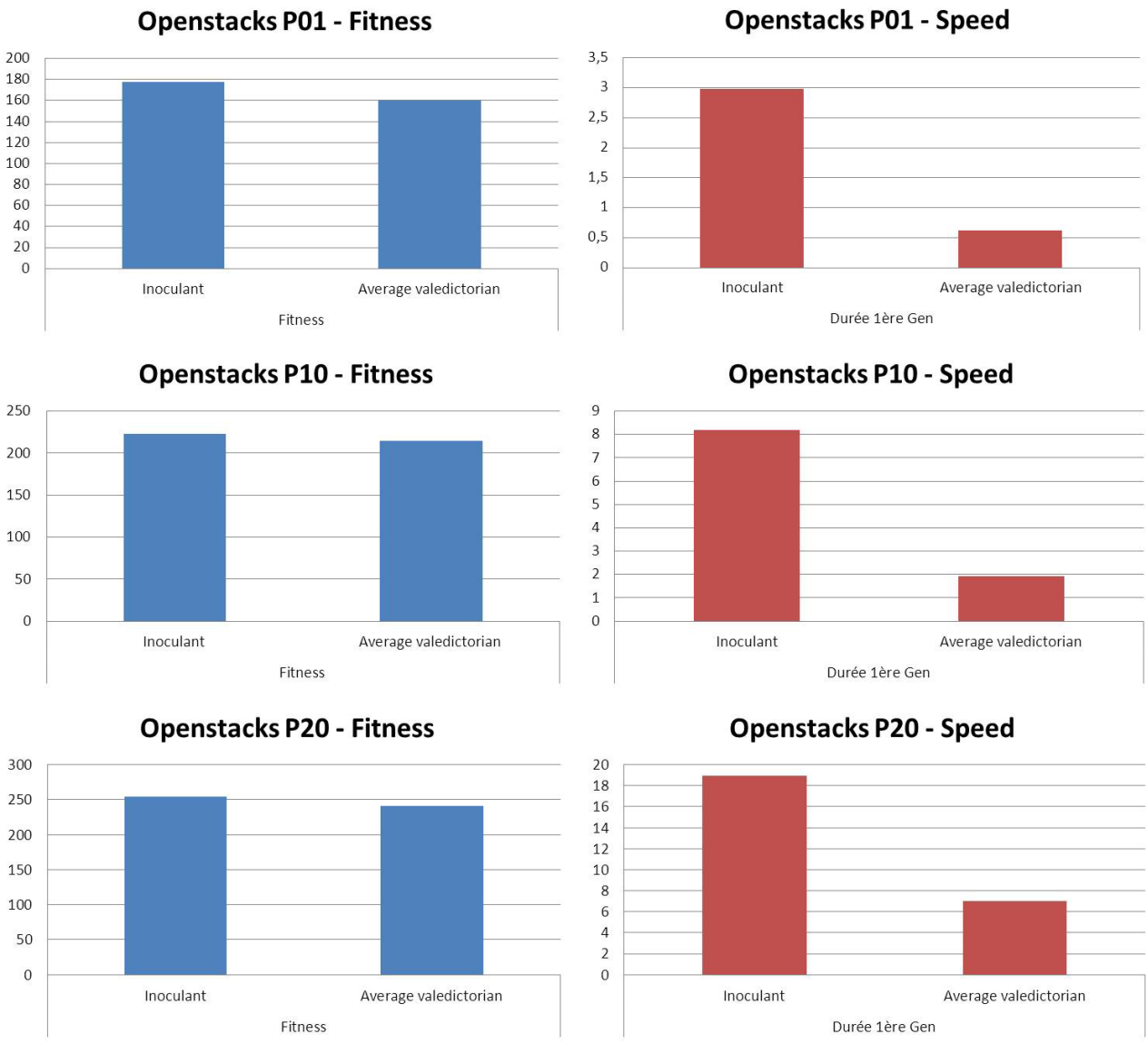

Figure 3.2: Basic inoculation performance on Openstacks

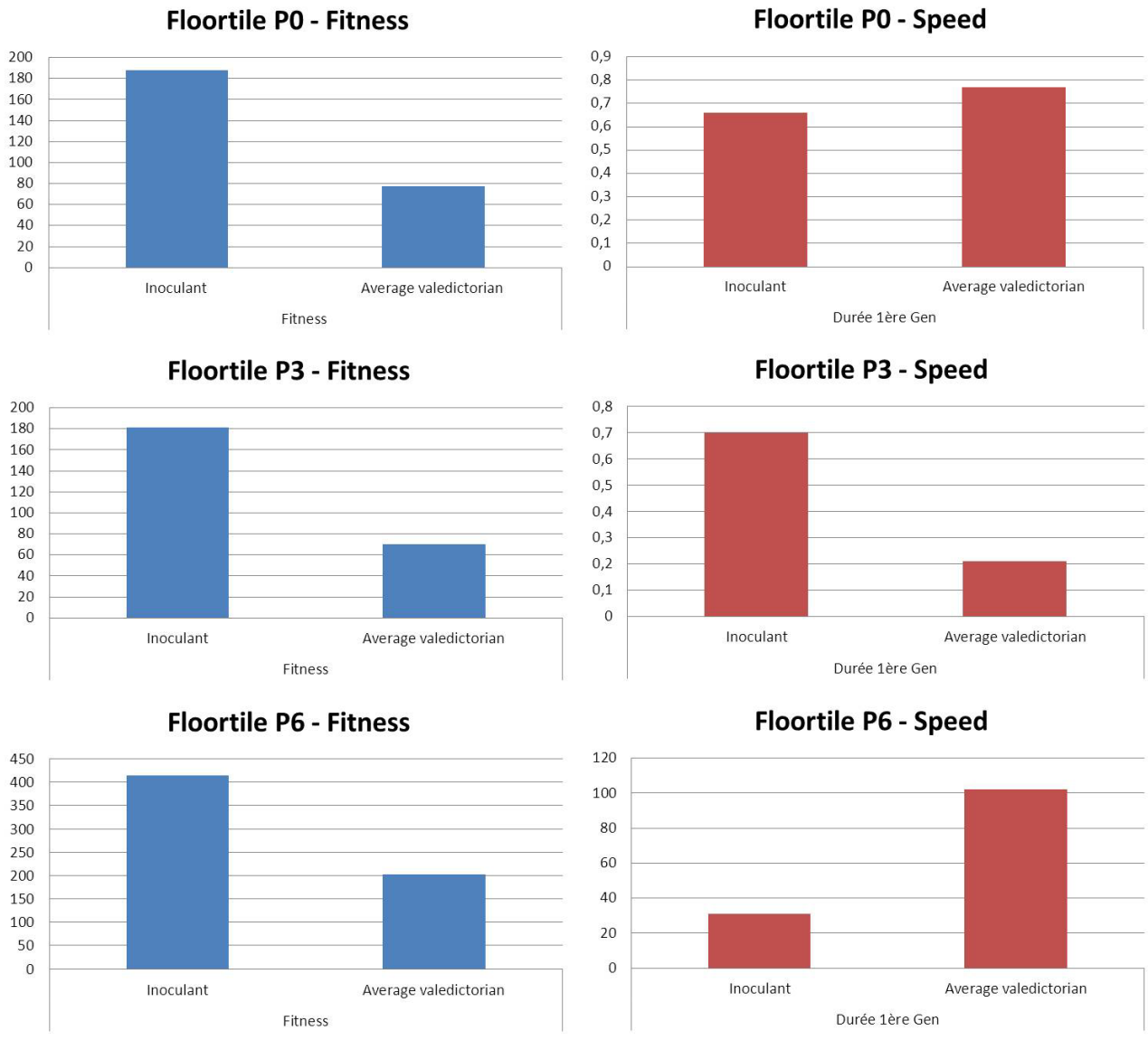

Figure 3.3: Basic inoculation performance on Floortile

4. INOCULATION THROUGH MASS SELECTION

4.1 MOTIVATION AND PRINCIPLE

While observing experimental logs of *Divide-and-Evolve*runs performed with or without inoculation, we noticed a number of things in addition to the conclusions reported above. First of all, in many cases, particularly with the elevators domain, the first challenge out algorithm needs to meet is to find one feasible individual with which it is bound to start the search with our parameter settings. To do so, it follows our loop based bmax estimation procedures which tend to increase bmax exponentially to stand a greater chance to find a way for Yahsp to build a feasible individual aorund the decompositions. This heuristic works in practice but what we noticed is that it is very costly in terms of speed as runs ending up with high values of bmax tended to be extremely slow. Zooming out it looked like a factor of risk: the run either got a "lucky" random seed leading to a feasible individual quickly with a low value of bmax allowing sufficient speed afterward or it got an "unlucky" one and got, purely and simply, lost because of unacceptable resulting speed. Another thing we noticed is that benefit of strongly increasing bmax was questionable as many runs, with lucky seeds and therefore low bmax values, could find feasible individuals without difficulty.

From these observations and from those reported above concerning the poor performance of Adam, we derived a new way initialize, or inoculate our evolutionary algorithm: instead of using Yashp to find starting point and instead of increasing bmax to find a first feasible individual, we will simply use additional sampling. In other terms, we will regenerate the initial population randomly with a fixed, low, value of bmax until it contains a feasible individual, which will serve as the inoculant. This very simple scheme should prove useful to reduce the risk of lost runs and increase the average speed at very little cost. The procedure is summarized in algorithm 5 and the following table:

Algorithm 5 Inoculation through mass selection

Require: $N, b_{max}fixed$ // initialization parameters
1: $b_{max} = b_{max}fixed$
2: $goodguys \leftarrow 0$
3: **while** $((goodguys = 0)AND(iter <= maxIter))$ **do**
4: $goodguys \leftarrow 0$
5: **for** $i \in [1..N]$ **do**
6: $pop[i] \leftarrow GenerateIndividual(K)$ // random individuals with K intermediate goals
7: **if** pop[i].isFeasible **then**
8: $goodguys++$

Inoculation through Mass Selection	
Substance provider	random sampling
Inoculant builder	none needed
Variation scheme	none
Composition scheme	1 inoculant + random

4.2 EXPERIMENTS

For all experiments, we use the same paramter settings reported above.

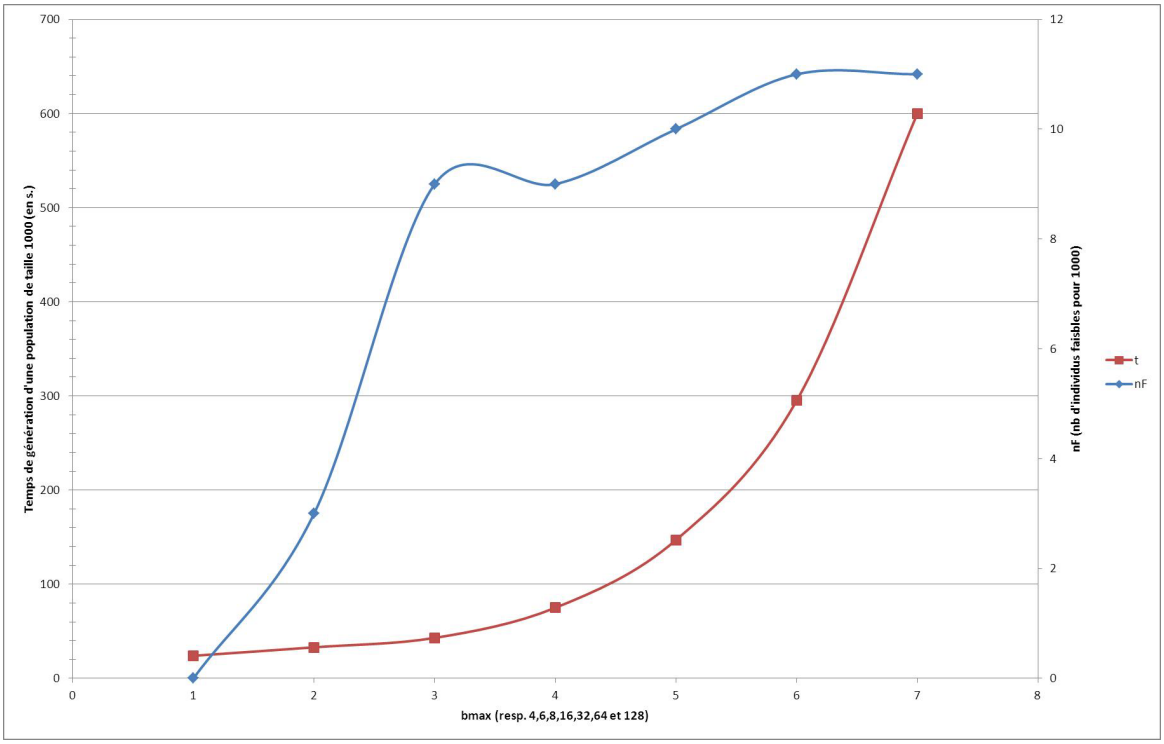

Figure 4.1: The consequences of increasing bmax

4.2.1 CONSEQUENCES OF INCREASING BMAX

In this first experiment, we formalize our observations that rapidly increasing bmax in search for a first feasible individual is not a viable strategy. To see that, we draw large, random, populations of size 1000 with increasing values of bmax in ranges typical of those observed in runs of *Divide-and-Evolve*performed on Elevators P01. As can be seen in figure 4.4, the number of feasible individuals practically ceases to increase significantly after a bmax value of 8! On the other hand, as one might have suspected, the time it takes to process individuals increases exponentially along with values of bmax as dictated by our estimation procedure. It is therefore clearly not beneficial to increase bmax senselessly: not only does it cost a lot in terms of speed but there is also simply almost no significant benefit to go beyond a certain bmax threshold, here 8.

4.2.2 FAILURE RATE

In this second experiment, we compare the proportions of lost runs between classic initialization and inoculation through mass selection as described above. A successful run is described as run finding a feasible individual to start the search without reaching a bmax value resulting in unreasonable slowness. For this experiment, the threshold for bmax for a run to be considered lost is 32.

As can be seen in figure 4.2, the failure rate is quite high with the classic procedure as run stands a 25% to 50% chance of leading nowhere. Our mass inoculation variant successfully removes that factor of risk on all attempted benchmarks.

4.2.3 START OVERHEAD OF SUCCESSFUL RUNS

In the third experiment, we try to look at the cost of this new procedure. Regenerating new random populations and evaluating them is indeed costly as it takes significant time and the start of search is consequently delayed. Figure 4.3 compares the times it takes to reach the first generation in the three cases of interest: classic

Figure 4.2: Proportion of lost runs on Elevators

initiliazation, classic inoculation and inoculation through mass selection. Although the addional sampling cost is visible with respect to classic initialization, inoculation through mass selection clearly removes the considerable overhead observed with Yahsp based inoculation.

4.2.4 AVERAGE SPEED OVER ALL RUNS

In this fourth and last experiment, we observe the benefit of mass inoculation in terms of average speed as opposed to classic initialization. Because it uses a fixed, low value of bmax, mass inoculation has higher resulting average speed than classic initialization which is based on a loop for the estimation of the proper value for bmax which results in a variety of values for bmax over the various runs/seeds, many being, as we saw, pointlessly higher than the manually set value. Of course this calls for a way of automatically coming up with this value but that is left for further work.

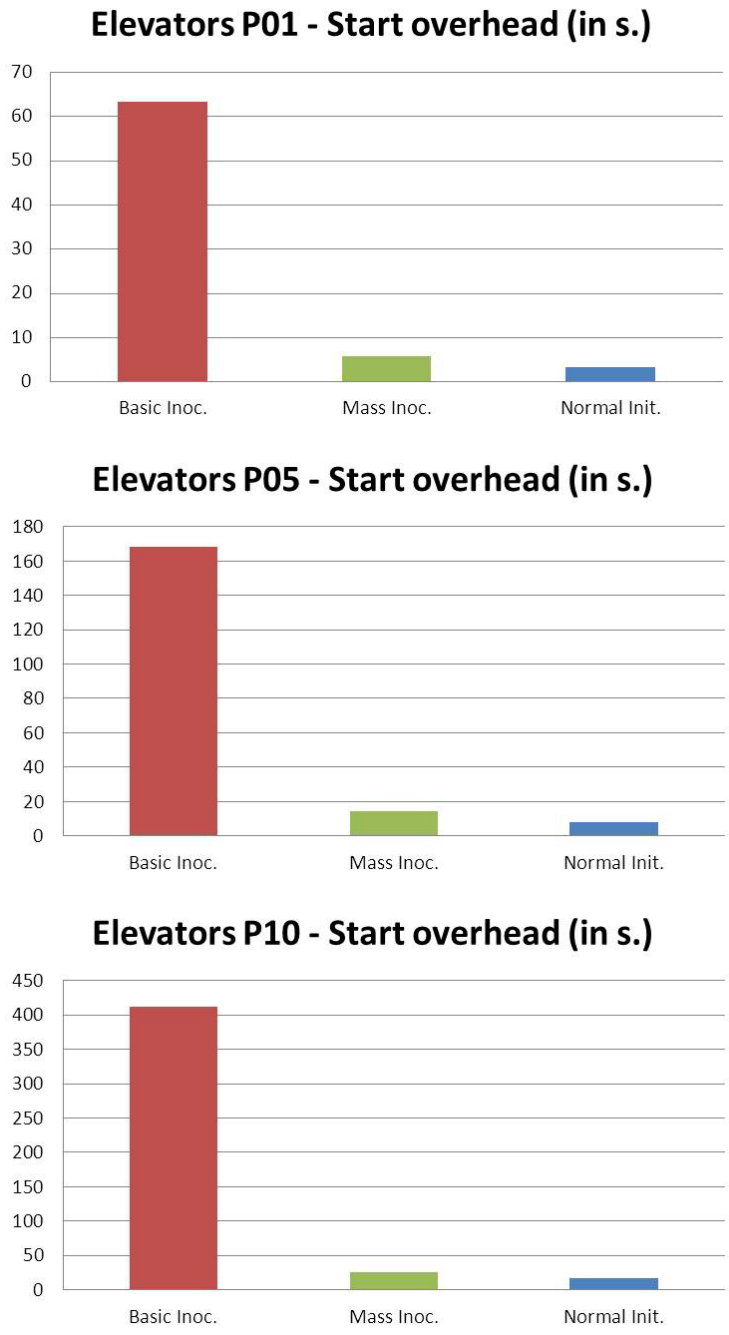

Figure 4.3: Cost of inoculation through mass selection in terms of start overhead

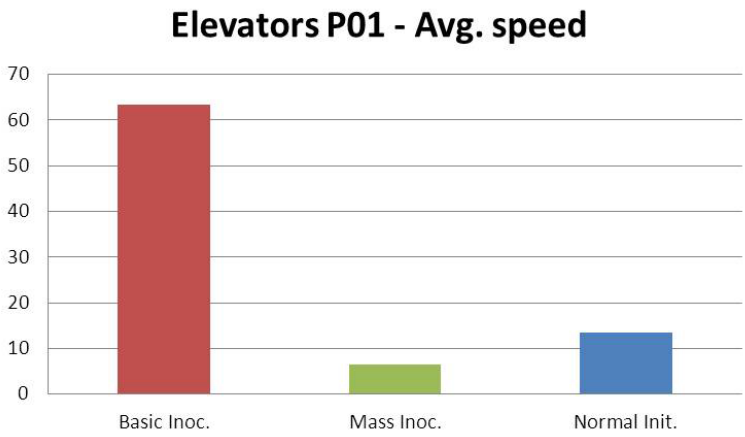

Figure 4.4: Benefit of inoculation through mass selection in terms of average speed

5. CONCLUSION

Inoculation can be a very good way to improve an evolutionary algorithm's performance in terms of speed, solution quality or robustness. There many ways to inoculate the population with relevant information and many ways to take advantage of that knowledge. In the context of *Divide-and-Evolve*, we tried two inoculation procedures. The first one, heuristically designed and based on our embedded planner, Yahsp, performs poorly: the resulting algorithm is slow, risky and produces low quality solutions. Experimental analysis however suggest that further work is required, particularly on the inoculant builder to tweak it and assess the actual benefit of using Yashp as a substance provider. Observation of experimental results additionally led us to imagine another, very simple, inoculation procedure based on extra sampling which successfully improves our algorithm on a number of benchmarks, in particular by reducing the stochastic risk of runs getting lost. Much further work is however needed to make this improvement robust and significant for all benchmark domains.

6. REFERENCES

[1] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. pages 38–46. Morgan Kaufmann Publishers, 1995.

[2] Jacques Bibai, Pierre Savéant, Marc Schoenauer, and Vincent Vidal. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *ICAPS*, pages 18–25, 2010.

[3] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *CoRR*, abs/1107.0052, 2011.

[4] Hillol Kargupta. Search, polynomial complexity, and the fast messy genetic algorithm. Technical report, 1995.

[5] Pedro Larrañaga and Jose A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

[6] Yann Semet and Marc Schoenauer. On the Benefits of Inoculation, an Example in Train Scheduling. In Mike Cattolico et al., editor, *GECCO-2006*, pages 1761–1768, Seattle, États-Unis, 2006. ACM Press.

[7] Patrick Surry. *A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms (Chapter 10)*. PhD thesis, 1998.

[8] Patrick D. Surry and Nicholas J. Radcliffe. Inoculation to initialise evolutionary search. In TerenceC. Fogarty, editor, *Evolutionary Computing*, volume 1143 of *Lecture Notes in Computer Science*, pages 269–285. Springer Berlin Heidelberg, 1996.