POSL: A Parallel-Oriented Solver Language

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

Alejandro Reyes Amaro

École doctorale : Sciences et technologies de l'information, et mathématiques
Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'Informatique de Nantes-Atlantique (LINA)

Directeur de thèse : M. Eric Monfroy, Professeur, Université de Nantes
Co-encadrant : M. Florian Richoux, Maître de Conferences, Université de Nantes

UNIVERSITÉ DE NANTES

Submitted: dd/mm/2016

Jury:

Président :	**M. Salvador ABREU**, *Professeur étranger*, Université d'Évora
Rapporteurs :	**M. Frédéric LARDEUX**, *Maître de conférences*, Université d'Angers
	M. Christophe LECOUTRE, *Professeur*, Université d'Artois
Examinateur :	**M. Arnaud LALLOUET**, *Chercheur industriel*, Huawei Technologies Ltd.

POSL: A Parallel-Oriented Solver Language

Short abstract:
The multi-core technology and massive parallel architectures are nowadays more accessible for a broad public through hardware like the Xeon Phi or GPU cards. This architecture strategy has been commonly adopted by processor manufacturers to stick with Moore's law. However, this new architecture implies new ways of designing and implementing algorithms to exploit their full potential. This is in particular true for constraint-based solvers dealing with combinatorial optimization problems.

Furthermore, the developing time needed to code parallel solvers is often underestimated. In fact, conceiving efficient algorithms to solve certain problems takes a considerable amount of time. In this thesis we present POSL, a Parallel-Oriented Solver Language for building solvers based on meta-heuristic, in order to solve Constraint Satisfaction Problems (CSP) in parallel. The main goal of this thesis is to obtain a system with which solvers can be easily built, reducing therefore their development effort, by proposing a mechanism of code reusing between solvers. It provides a mechanism to implement solver-independent communication strategies. We also present a detailed analysis of the results obtained when solving some CSPs. The goal is not to outperform the state of the art in terms of efficiency, but showing that it is possible to rapidly prototyping with POSL in order to experiment different communication strategies.

Keywords: Constraint satisfaction, meta-heuristics, parallel, inter-process communication, language.

Contents

Part I

Study and evaluation of POSL

1

Experiments design and results

In this Chapter I expose all details about the evaluation process of POSL*, i.e., all experiments I perform. For each benchmark, I explain used strategies in the evaluation process and the used environments were the runs were performed (Curiosiphi server). I describe all the experiments and I expose a complete analysis of the obtained result.*

Contents

In this chapter I illustrate and analyze the versatility of POSL studying different ways to solve constraint problems based on local search meta-heuristics. I have chosen the *Social Golfers Problem*, the *N-Queens Problem*, the *Costas Array Problem* and the *Golomb Ruler Problem* as benchmarks since they are challenging yet differently structured problems. In this chapter I present formally each benchmark, I explain the structure of POSL's solvers that I have generated for experiments and present a detailed analysis of obtained results.

First results using POSL to solve constraint problems were published in [6] were we used POSL to solve the *Social Golfers Problem* and to study some communication strategies. It was the first version of POSL, therefore it was able to solve only relatively easy instances. However, results suggested that the communication can play an important role if we are able to find the proper communication strategy.

1.1 Methodology

Some terms are necessary to be defined for simplification sake. They are the *sequential environment* and the *parallel environment*, which are the description of the computation resources used for experimentation. Experiments[i] were performed on an Intel® Xeon™ E5-2680 v2, 10×4 cores, 2.80GHz. This server is called Curiosiphi and is located at *Laboratoire d'Informatique de Nantes Atlantique*, at the University of Nantes.

Definition 1 *We say that we launch an experiment using a* ***sequential environment*** *if we execute a solver set into a single process of* Curiosiphi.

Definition 2 *We say that we launch an experiment using a* ***parallel environment*** *if we execute a solver set in parallel (multi-walk) using the maximum of available processes in* Curiosiphi.

With the aim of being as exhaustive as possible in the experimentation process, a methodology based on four stages is proposed:

1. **Algorithm selection** In this stage some experiments are launched to ensure choosing the right computation modules, and the right design of the abstract solver. Experiments are performed using the sequential environment, and the following statistical analysis is performed: A set of 30 runs for each setup are performed, and used to a) build box-plot diagrams and bars graphs with some additional information about winner

[i] POSL source code is available on GitHub:https://github.com/alejandro-reyesamaro/POSL

solvers, presented in Appendixes A, B, C and D; b) compute means and standard deviation for run-times and iterations, showed in tables presented in this chapter, in columns labeled **T** (run-time in seconds), **It.** (number of iterations), **T(sd)** and **It.(sd)** (their respective standard deviations). In some tables, the column labeled **% success** indicates the percentage of solvers finding a solution before reaching a time–out (5 minutes).

2. **Algorithm evaluation in the parallel environment** The selected algorithm is launched using the parallel environment. It is performed a similar statistical analysis to the one described in the previews stage, and results are compared.

3. **Communication strategies selection** After a detailed study of the search process and the behavior of the designed solver sets, some changes in the solver set are proposed in order to design a communication strategy:

   - replacing some computation modules for others based on the originals, but with some modifications according to the new demands of the proposed communication strategy;
   - adding some communication modules depending on the information that we intend to share;
   - a new abstract solver is coded, whose modifications are the strictly necessary to incorporate communication modules;
   - the structure of the communication is designed in order to chose the right communication operators.

4. **Communication strategy evaluation** The designed communication strategy is launched suing the parallel environment, and a statistical analysis is performed. Communication strategies are compared each others based on obtained results in order to select the right one. These results are also compared to those obtained during the stage 2., to be able to draw conclusions about the success of the cooperative approach.

It is important to point out that POSL is not designed to obtain the best results in terms of performance, but to give the possibility of rapidly prototyping and studying different cooperative or non cooperative search strategies.

1.2 A dynamic configuration exchange strategy (*Social Golfers*)

In this section I present the performed study using *Social Golfers Problem* (*SGP*) as a benchmark. The communication strategy analyzed here consists in applying a mechanism

of cost descending acceleration, exchanging the current configuration between two solvers with different characteristics. Final obtained results show that this communication strategy works pretty well for this problem.

1.2.1 Problem definition

The *Social Golfers Problem* (*SGP*) consists in scheduling $g \times p$ golfers into g groups of p players every week for w weeks, such that two players play in the same group at most once. An instance of this problem can be represented by the triple $g - p - w$. This problem, and other closely related problems, arise in many practical applications such as encoding, encryption, and covering problems [7]. Its structure is very attractive, because it is very similar to other problems, like *Kirkman's Schoolgirl Problem* and the *Steiner Triple System*.

The cost function for this benchmark was implemented making an efficient use of the stored information about the cost of the previews configuration. Using integers to work with bit-flags, a table to store the information about the partners of each player in each week can be filled in $O(p^2 \cdot g \cdot w)$. So, if a configuration has $n = (p \cdot g \cdot w)$ elements, this table can be filled in $O(p \cdot n)$. This table is filled from scratch only one time in the search process (I explain in the next section why). Then, every cost of a new configuration, is calculated based on this information and the performed changes between the new configuration and the stored one. This relative cost is calculated in $O(c \cdot g)$, where c is the number of performed changed in the new configuration with respect to the stored one.

1.2.2 Experiment design and results

Here, I present the abstract solver designed for this problem as well as concrete computation modules composing the different solvers I have tested:

1. Generation abstract module I:

   I_{BP}: Returns a random configuration s, respecting the structure of the problem, *i.e.*, the configuration is a set of w permutations of the vector $[1..n]$, where $n = g \times p$.

2. Neighborhood abstract modules V:

   V_{std}: Given a configuration, returns the neighborhood $\mathcal{V}(s)$ swapping players among groups.

Algorithm 1: Simple solvers for *SGP*

```
abstract solver as_simple                                   // ITR → number of iterations
computation : I, V, S, A
begin
    [↺ (ITR < K1) I ↦ [↺ (ITR % K2) [V ↦ S ↦ A] ] ]
end
solver SOLVER_Std implements as_simple
    computation : I_BP, V_std, S_best, A_AI
solver SOLVER_AS implements as_simple
    computation : I_BP, V_BAS, S_best, A_AI
```

V_{BAS}: Given a configuration, returns the neighborhood $\mathcal{V}(s)$ swapping the most culprit player with other players from the same week. It is based on the *Adaptive Search* algorithm.

$V_{BP}(p)$: Given a configuration, returns the neighborhood $\mathcal{V}(s)$ by swapping the most culprit player with other players in the same week, for all p randomly selected weeks.

3. Selection abstract modules S:

   S_{first}: Given a neighborhood, selects the first configuration $s' \in V(s)$ improving the current cost and returns it together with the current one into the pair (s', s).

   S_{best}: Given a neighborhood, selects the best configuration $s' \in V(s)$ improving the current cost and returns it together with the current one into the pair (s', s).

   S_{rand}: Given a neighborhood, selects randomly a configuration $s' \in V(s)$ and returns it together with the current one, into the pair (s', s).

4. Acceptance abstract module A:

   A_{AI}: Given a pair (s', s), returns always the configuration s'

These concrete modules are very useful and can be reused to solve tournament-like problems like *Sports Tournament Scheduling*, *Kirkman's Schoolgirl* and the *Steiner Triple System* problems.

In a first stage of the experiments I use the operator-based language provided by POSL to build and test many different non communicating strategies. The goal is to select the best concrete modules to run tests performing communication. A very first experiment was performed to select the best neighborhood function to solve the problem, comparing a basic solver using V_{std}; a new solver using V_{BAS}; and a combination of V_{std} and V_{BAS} by applying the operator (ρ), already introduced in the previous chapter. Algorithms 1 and 2 present solvers for each case, respectively.

Solver	T	T(sd)	It.	It.(sd)
SOLVER$_{AS}$	**1.06**	0.79	352	268
SOLVER$_{rho}$	41.53	26.00	147	72
SOLVER$_{Std}$	87.90	41.96	146	58

Table 1.1: *Social Golfers*: Instance 10–10–3 in parallel

Algorithm 2: Solvers combining neighborhood functions using operator *RHO*

abstract solver *as_rho* // ITR → number of iterations
computation : I, V_1, V_2, S, A
begin
$[(\circlearrowleft)\ (\text{ITR} < K_1)\ I\ (\mapsto)\ [(\circlearrowleft)\ (\text{ITR}\ \%\ K_2)\ [[V_1\ (\rho)\ V_2]\ (\mapsto)\ S\ (\mapsto)\ A]\ ]\ ]$
end
solver SOLVER$_{rho}$ **implements** as_rho
computation : $I_{BP}, V_{std}, V_{BAS}, S_{best}, A_{AI}$

Results in Table 1.1 are not surprising. The neighborhood module V_{BAS} is based on the *Adaptive Search* algorithm, which has shown very good results [8]. It selects the most culprit variable (i.e., a player), that is, the variable the most responsible for constraints violation. Then, it permutes this variable value with the value of each other variable, in all groups and all weeks. Each permutation gives a neighbor of the current configuration. V_{Std} uses no additional information, so it performs every possible swap between two players in different groups, every week. It means that this neighborhood is $g \times p$ times bigger than the previous one, with g the number of groups and p the number of players per group. It allows for more organized search because the set of neighbors is pseudo-deterministic, i.e., the construction criteria is always the same but the order of the configuration is random. On the other hand, *Adaptive Search* neighborhood function takes random decisions more frequently, and the order of the configurations is random as well. We also tested a solver with combining these modules using the (ρ) operators. This operator executes its first or second parameter depending on a given probability ρ. This combination spent more time searching the best configuration among the neighborhood, although with a lower number of iterations than V_{BAS}. The V_{BAS} neighborhood function being clearly faster, we have chosen it for our experiments, even if it shown a more spread standard deviation: 0.75 for SOLVER$_{AS}$ versus 0.62 for SOLVER$_{Std}$, considering the ratio $\frac{T(sd)}{T}$.

***

Once the neighborhood computation module has been selected, I have focused the experiment on choosing the best *selection* computation module. Solvers mentioned above were too slow to solve instances of the problem with more than three weeks: they were very often trapped into local minima. For that reason, another solver implementing the abstract solver described

Instance	Best improvement				First improvement			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
5–3–7	0.45	0.70	406	726	0.23	0.14	142	67
8–4–7	0.37	0.11	68	13	0.28	0.07	93	13
9–4–8	0.87	0.13	95	17	0.60	0.16	139	18

Table 1.2: *Social Golfers*: comparing selection functions in parallel

in Algorithm 3 have been created, using V_{BAS} and combining S_{best} and S_{rand}: it tries a number of times to improve the cost, and if it is not possible, it picks a random neighbor for the next iteration. We also compared the S_{first} and S_{best} selection modules. The computation module S_{best} selects the best configuration inside the neighborhood. It did not only spend more time searching a better configuration, but also is was more sensitive to become trapped into local minima. The second computation module S_{first} selects the first configuration inside the neighborhood improving the current cost. Using this module, solvers favor exploration over intensification and of course spend clearly less time searching into the neighborhood.

Algorithm 3: Solver for *SGP* to scape from local minima

```
abstract solver as_eager                          // ITR → number of iterations
computation : I, V, S1, S2, A
begin
    [(↺) (ITR < K1) I (↦) [(↺) (ITR % K2) [V (↦) [S1 (?)SCI<K3 S2] (↦) A] ] ]
end
solver SOLVERbest implements as_eager
    computation : IBP, Vstd, VBAS, Sbest, Srand, AAI
solver SOLVERfirst implements as_eager
    computation : IBP, Vstd, VBAS, Sfirst, Srand, AAI
```

Instance	T	T(sd)	It.	It.(sd)
5–3–7	1.25	1.05	2,907	2,414
8–4–7	0.60	0.33	338	171
9–4–8	1.04	0.72	346	193

Table 1.3: *Social Golfers*: a single sequential solver using first improvement

Tables 1.2 and 1.3 present results of this experiment, showing that a local exploration-oriented strategy is better for the *SGP*. If we compare results of Tables 1.2 1.3 with respect to the standard deviation, we can some gains in robustness with parallelism. The spread in the running times and iterations for the instance 5–3–7 is 24% lower (0.84 sequentially versus 0.60 in parallel), for 8–4–7 is 30% lower (0.55 sequentially versus 0.25 in parallel) and for 9–4–8 (the hardest one) is 43% lower (0.69 sequentially versus 0.26 in parallel), using the same ratio $\frac{T(sd)}{T}$).

***

The conclusion of the last experiment was that the best solver to solve *SGP* using POSL is the one using a neighborhood computation module based on *Adaptive Search* algorithm (V_{BAS}) and a selection computation module selecting the first configuration improving the cost. Using this solver as a base, the next step was to design a simple communication strategy where the shared information is the current configuration. Algorithms 4 and 5 show that the communication is performed while applying the acceptance criterion of the new configuration for the next iteration. Here, receiver solvers receive a configuration from a sender solver, and match it with their current configuration. Then, the configuration with the lowest global cost is selected. This operation is coded using the *minimum* operator (m) in Algorithm 5. This way, the receiver solver continues the search from a more promising place into de search space. Different communication strategies were designed, either executing a full connected solvers set, or a tuned combination of connected and unconnected solvers. Between connected solvers, two different connections operations were applied: connecting each sender solver with one receiver solver (one to one), or connecting each sender solver with all receiver solvers (one to N). The code for the different communication strategies are presented in Algorithms 6 to 11.

Algorithm 4: Communicating abstract solver for *SGP* (sender)

```
abstract solver as_eager_sender                    // Itr → number of iterations
computation : I, V, S1, S2, A                      // Sci → number of iterations with the same cost
begin
    [(↻) (Itr < K1)  I (↦)  [(↻) (Itr % K2)  [V (↦)  [S1 (?)Sci<K3 S2]  (↦) (|A|)^o ]  ]
end
solver Solver_sender implements as_eager_sender
    computation : I_BP, V_BAS, S_first, S_rand, A_AI
```

Algorithm 5: Communicating abstract solver for *SGP* (receiver)

```
abstract solver as_eager_receiver                  // Itr → number of iterations
computation : I, V, S1, S2, A                      // Sci → number of iterations with the same cost
communication : C.M.
begin
    [(↻) (Itr < K1)
        I (↦)  [(↻) (Itr % K2)  V (↦)  [S1 (?)Sci<K3 S2]  (↦)  [A (m) C.M.]  ]
    ]
end
solver Solver_receiver implements as_eager_receiver
    computation : I_BP, V_BAS, S_first, S_rand, A_AI
    communication : CM_last
```

In Algorithm 5, the abstract communication module $C.M.$ was instantiated with the concrete communication module CM_{last}, which takes into account the last received configuration at

Algorithm 6: Communication strategy one to one 100%

$[\text{SOLVER}_{sender} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{receiver} \cdot C.M.]\, 20;$

Algorithm 7: Communication strategy one to N 100%

$[\text{SOLVER}_{sender} \cdot A(20)] \boxed{\rightsquigarrow} [\text{SOLVER}_{receiver} \cdot C.M.(20)]\,;$

the time of its execution.

Each time a POSL meta-solver is launched, many independent search solvers are executed. We call "good" configuration a configuration with the lowest cost within the current configuration neighborhood and with a cost strictly lesser than the current one. Once a good configuration is found in a sender solver, it is transmitted to the receiver one. At this moment, if the information is accepted, there are some solvers searching in the same subset of the search space, and the search process becomes more exploitation–oriented. This can be problematic if this process makes solvers converging too often towards local minima. In that case, we waste more than one solver trapped into a local minima: we waste all solvers that have been attracted to this part of the search space because of communications. This phenomenon is avoided through a simple (but effective) play: if a solver is not able to find a better configuration inside the neighborhood (executing S_{first}), it selects a random one at the next iteration (executing S_{rand}).

In all Algorithms in this section, three parameter can be found: 1. K_1: the maximum number of *restarts*, 2. K_2: the maximum number of iterations in each *restart*, and K_3: the maximum number of iterations with the same current cost. 3.

After the selection of the proper modules to study different communication strategies, I proceeded to tune these parameter. Only a few runs were necessaries to conclude that the mechanism of using the computation module S_{rand} to scape from local minima was enough. For that reason, since the solver never perform restarts, the parameter K_1 was irrelevant. So the reader can assume $K_1 = 1$ for every experiment.

With the certainty that solvers do not performs restarts during the search process, I select the same value for $K_2 = 5000$ in order to be able to use the same abstract solver for all instances.

Finally, in the tuning process of K_3, I notice only slightly differences between using the values 5, 10, and 15. So I decided to use $K_3 = 5$.

Algorithm 8: Communication strategy one to one 50%

$[\text{SOLVER}_{sender} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{receiver} \cdot C.M.]\, 10;$
$[\text{SOLVER}_{first}]\, 20;$

Algorithm 9: Communication strategy one to N 50%

$[\text{SOLVER}_{sender} \cdot A(10)] \boxed{\rightsquigarrow} [\text{SOLVER}_{receiver} \cdot C.M.(10)]$;
$[\text{SOLVER}_{first}]\, 20$;

Algorithm 10: Communication strategy one to one 25%

$[\text{SOLVER}_{sender} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{receiver} \cdot C.M.]\, 5$;
$[\text{SOLVER}_{first}]\, 30$;

Algorithm 11: Communication strategy one to N 25%

$[\text{SOLVER}_{sender} \cdot A(5)] \boxed{\rightsquigarrow} [\text{SOLVER}_{receiver} \cdot C.M.(5)]$;
$[\text{SOLVER}_{first}]\, 30$;

Instance	Communication 1 to 1				Communication 1 to N			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
5–3–7	0.20	0.20	165	110	0.20	0.17	144	108
8–4–7	0.27	0.09	88	28	0.24	0.05	95	12
9–4–8	0.52	0.14	117	25	0.55	0.14	126	20

Table 1.4: *Social Golfers*: 100% of communicating solvers

Instance	Communication 1 to 1				Communication 1 to N			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
5–3–7	0.18	0.13	125	88	0.17	0.12	139	81
8–4–7	0.21	0.06	89	18	0.22	0.06	90	20
9–4–8	0.49	0.11	119	24	0.51	0.15	124	21

Table 1.5: *Social Golfers*: 50% of communicating solvers

Instance	Communication 1 to 1				Communication 1 to N			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
5–3–7	0.22	0.20	181	130	0.23	0.16	143	80
8–4–7	0.24	0.08	95	22	0.29	0.09	93	12
9–4–8	0.55	0.14	134	21	0.55	0.11	130	20

Table 1.6: *Social Golfers*: 25% of communicating solvers

This communication strategy produces some gain in terms of runtime (Table 1.2 with respect to Tables 1.4, 1.5 and 1.6). Having many solvers searching in different places of the search space, the probability that one of them reaches a promising place is higher. Then, when a solver finds a good configuration, it can be communicated, and receiving the help of one or more solvers in order to find the solution. Using this strategy, the spread in the running times and iterations was reduced for the instance 9–4–8 (0.22 using communication one to one and 50% of communication solvers), but not for instances 5–3–7 and 8–4–7 (0.70 using communication one to N and 50% of communicating solvers, and 0.28 using communication one to one and 50% of communicating solvers, respectively).

Other two strategies were analyzed in the resolution of this problem, with no success, both based on the sub-division of the work by weeks, i.e., solvers trying to improve a configuration only working with one or some weeks. To this end two strategies were designed:

A **Circular strategy:** K solvers try to improve a configuration during a during a number of iteration, only working on one week. When no improvement is obtained, the current configuration is communicated to the next solver (circularly), which tries to do the same working on the next week (see Figure 1.1a).

   This strategy does not show better results than previews strategies. The reason is because, although the communication in POSL is asynchronous, most of the times solvers were trapped waiting for a configuration coming from its neighbor solver.

B **Dichotomy strategy:** Solvers are divided by levels. Solvers in level 1, only work on one week, solvers on level 2, only work on 2 consecutive weeks, and so on, until the solver that works on all (except the first one) weeks. Solvers in level 1 improve a configuration during some number of iteration, then this configuration is sent to the corresponding solver. Solvers in level 2 do the same, but working on weeks k to $k+1$. It means that it receives configurations from the solver working on week k and from the solver working on week $k+1$, and sends its configuration to the corresponding solver working on weeks k to $k+3$; and so on. The solver in the last level works on all (except the first one) weeks and receive configuration from the solver working on weeks 2 to $w/2$ and from the solver working on weeks $w/2+1$ to w (see Figure1.1b). We tested this strategy with all possible levels.

   The goal of this strategy was testing if focused searches rapidly communicated can help at the beginning of the search. However, The failure of this strategy is in the fact that most of the time the sent information arrives to late from the bottom to the top.

***

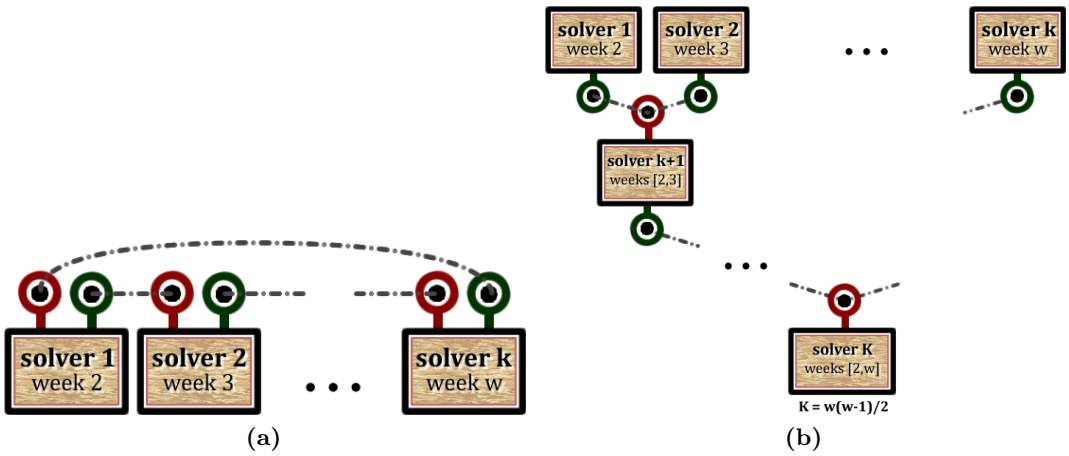


Figure 1.1: Unsuccessful communication strategies to solve *SGP*

One last experiment using this benchmark was implementing a communication strategy which applies a mechanism of cost descending acceleration, exchanging the current configuration between two solvers with different characteristics. Results show that this communication strategy works pretty well for this problem.

For this strategy, new solvers were built reusing same modules used for the communication strategies exposed before, and another different neighborhood computation module: $V_{BP}(p)$, which given a configuration, returns the neighborhood $\mathcal{V}(s)$ by swapping the culprit player chosen for all p randomly selected weeks with other players in the same week. This new solver was called *companion solver*, and it descends quicker the cost of its current solution at the beginning because its neighborhood generates less values, but the convergence is slower and yet not sure. It was combined with the solver similar to the one used for the communication strategies exposed before. It was called *standard solver*, and converges in a stable way to the solution. So, the companion solver uses the same neighborhood function that the standard solver, but parametrized in such a way that it builds neighbors only swapping players among two weeks.

The idea of the communication strategy is to communicate a configuration from the companion solver to the standard solver, to be able to continue the search from a more promising place into de search space. After some iterations, is the standard solver who sends its configuration to the companion solver. The companion solver takes this received configuration and starts its search from it and finds quickly a much better configuration to send to the standard solver again. To force the companion solver to take the received configurations over its own, we use the *not null* operator together with the communication module $C.M.$ (Algorithm 13). This process is repeated until a solution is found.

Figure 1.2 shows standard solver's run versus companion solver's run. In this chart we can see that, at the beginning of the run, found configurations by the companion solver

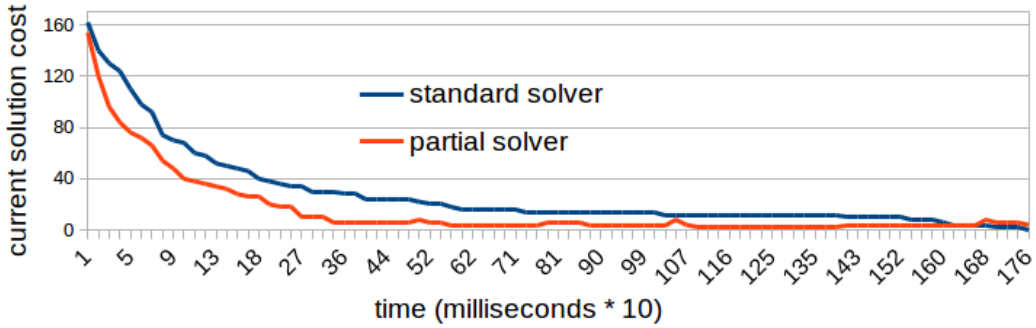


Figure 1.2: Companion solver vs. standard solver (solving *Social Golfers Problem*)

have costs significantly lower than those found by the standard solver. At the 60-th millisecond the standard solver current configuration has cost 123, and the companion solver's one, 76. So for example, the communication at this time, can accelerate the process significantly.

Algorithm 12: Standard solver for *SGP*

abstract solver *as_standard*
computation : I, V, S_1, S_2, A
communication : $C.M.$
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ \ V \ (\mapsto) \ [S_1 \ (?)_{\text{SCI}\%K_1} \ S_1] \ (\mapsto) \ [C.M. \ (m) \ (\!|A|\!)^d] \ ]$
end
solver SOLVER$_{standard}$ **implements** as_standard
computation : $I_{BP}, V_{BAS}, S_{first}, S_{rand}, A_{AI}$
communication : CM_{last}

Algorithm 13: Companion solver for *SGP*

abstract solver *as_companion*
computation : I, V, S_1, S_2, A
communication : $C.M.$
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ \ V \ (\mapsto) \ [S_1 \ (?)_{\text{SCI}\%K_1} \ S_2] \ (\mapsto) \ [C.M. \ (\vee) \ (\!|A|\!)^d] \ ]$
end
solver SOLVER$_{companion}$ **implements** as_companion
computation : $I_{BP}, V_{BP}(2), S_{first}, S_{rand}, A_{AI}$
communication : CM_{last}

We also design different communication strategies, combining connected and unconnected solvers in different percentages, and applying two different communication operators: one to one and one to N.

This strategy produces some gain in terms of runtime as we can see in Tables 1.7 and 1.8 with respect to Table 1.2. It produces also more robust results in terms of runtime. The spread of results in iterations show higher variances, because there are included also results of companion solvers, which performs many times more iterations that the standard solvers. The

Instance	Comm. one to N				(Comm. one to N)/2				(Comm. one to N)/4			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
5–3–7	0.14	0.08	102	53	0.14	0.07	97	73	0.12	0.08	175	162
8–4–7	0.30	0.13	101	24	0.22	0.06	92	29	0.22	0.06	88	45
9–4–8	0.55	0.15	125	20	0.53	0.14	107	20	0.40	0.14	101	70

Table 1.7: Companion communication strategy with communication one to N

Instance	Comm. one to one (100%)				Comm. one to one (50%)				Comm. one to one (25%)			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
5–3–7	0.10	0.05	98	75	**0.08**	0.04	139	122	0.11	0.05	190	142
8–4–7	**0.14**	0.05	100	64	0.22	0.06	119	74	0.21	0.5	101	64
9–4–8	0.37	0.14	86	65	**0.36**	0.12	144	92	0.45	0.11	150	96

Table 1.8: companion communication strategy with communication one to N

percentage of the receiver solvers that were able to find the solution before the others did, was significant (see Appendix A, Figures A.5, A.6 and A.7). That shows that the communication played an important role during the search, despite inter–process communication's overheads (reception, information interpretation, making decisions, etc).

The code for the communication strategy of 100% of communicating solvers is presented in Algorithm 14 and for 50% of communicating solvers in Algorithm 15.

Algorithm 14: Companion communication strategy 100% communication

$[\text{SOLVER}_{companion} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{standard} \cdot C.M.]\, 20;$
$[\text{SOLVER}_{standard} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{companion} \cdot C.M.]\, 20;$

Algorithm 15: Companion communication strategy 50% communication

$[\text{SOLVER}_{companion} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{standard} \cdot C.M.]\, 10;$
$[\text{SOLVER}_{standard} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{companion} \cdot C.M.]\, 10;$
$[\text{SOLVER}_{first}]\, 20;$

1.3 A cyclic communication strategy (*N-Queens*)

In this section I present the performed study using *N-Queens Problem* (*NQP*) as a benchmark. The communication strategy analyzed here consists in exchanging cyclically the configuration between solvers using different neighborhood functions, in order to accelerate the process of generating very promising configurations. Final obtained results show that this communication strategy works pretty well for some instances of this problem.

1.3.1 Problem definition

The *N-Queens Problem* (*NQP*) asks how to place N queens on a chess board so that none of them can hit any other in one move. This problem was introduced in 1848 by the chess player Max Bezzelas as the *8-queen problem*, and years latter it was generalized as *N-queen problem* by Franz Nauck. Since then many mathematicians, including Gauss, have worked on this problem. It finds a lot of applications, e.g., parallel memory storage schemes, traffic control, deadlock prevention, neural networks, constraint satisfaction problems, among others [9]. Some studies suggest that the number of solution grows exponentially with the number of queens (N), but local search methods have been shown very good results for this problem [10]. For that reason we tested some communication strategies using POSL, to solve a problem relatively easy to solve using non communication strategies.

The cost function for this benchmark was implemented in C++ based on the current implementation of *Adaptive Search*[ii].

1.3.2 Experiments and results

To handle this problem, some modules used for the *Social Golfers Problem* have been reused: the selection computation modules S_{first} and S_{best}, and the acceptance computation module A_{AI}. It was used a simple abstract solver presented in Algorithm 16:

Algorithm 16: Abstract solver for *NQP*

abstract solver *as_simple* // ITR → number of iterations
computation : I, V, S, A // SCI → number of iterations with the same cost
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ V \ (\mapsto) \ S \ (\mapsto) \ A \ ]$
end
solver SOLVER$_{as}$ **implements** as_simple
computation : $I_{perm}, V_{AS}, S_{first}, A_{AI}$
solver SOLVER$_{selective}$ **implements** as_simple
computation : $I_{perm}, V_{PAS}(p), S_{first}, A_{AI}$

Solvers used for the experiments without communications, are presented in Algorithm 16, where the abstract solver is instantiated in the solver SOLVER$_{as}$ with the neighborhood computation module V_{AS}, which given a configuration, returns a neighborhood $V(s)$ swapping the variable which contributes the most to the cost, with all others. This solver was able to find solutions but taking too much time (a minute for 6000-queens, for example). For that

[ii] It is based on the code from Daniel Díaz available at https://sourceforge.net/projects/adaptivesearch/

Instance	Sequential				Parallel			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
250	0.29	0.072	8,898	2,158	0.19	0.003	4,139	913
500	0.35	0.087	4,203	998	0.24	0.036	2,675	366
1000	0.35	0.126	2,766	445	0.30	0.037	2,102	222
3000	1.50	0.138	2,191	77	1.33	0.055	2,168	71
6000	4.71	0.183	3,339	51	4.57	0.123	3,323	43

Table 1.9: Results for *NQP* (sequential and parallel without communication)

reason it was implemented a neighborhood computation module $S_{PAS}(p)$ which performs the same algorithm of S_{AS}, but instead of generating neighbors swapping the most costly variable with all others, it is swapped only with a percentage of rest of variables. Solver $\text{SOLVER}_{selective}$ instantiates the abstract solver with this computation module, showing much better results than SOLVER_{as}.

Table 1.9 presents results of sequential and parallel runs, using $\text{SOLVER}_{selective}$ with a tunned value of $p = 2.5$. Results show that the improvement of the parallel scheme using POSL is not big. While the number of solutions of this problem is only known for the very small value of $N = 23$, studies suggest that the number of solutions grows exponentially with N. It implies that as the problem grows in order, it becomes easier to solve through local search methods. The behavior of POSL solving this problem matches with this hypothesis: the search process solving larger instances is more stable and the convergence is direct. In a well spread search space with a lot of solutions, the parallelism using 40 cores do not provide a lot of improvement. In that sense, as a future work, experiments using POSL to solve *NQP* in parallel with more cores are planned.

***

In order to test the cooperative approach with this problem, a first and simple experiment was performed. Using the previous defined abstract solver, communicating solvers were built, to create a simple communication strategy in which the shared information is the current configuration, and it is communicated in one sense (from sender solvers to receivers). Algorithms 17 and 18 show that the communication is performed while applying the acceptance criterion. We design different communication strategies:

- a set of sender solvers sending information to receiver solvers, using operator one to one (see see Algorithm 19) and operator one to N (see Algorithm 20 with $K = 1$)
- some sets of sender solvers sending information to receiver solvers, using operator one to N (see see Algorithm 20), with $K \in \{2, 4\}$

Instance	Communication 1-1			
	T	T(sd)	It.	It.(sd)
250	0.18	0.040	3,433	697
500	0.25	0.047	2,216	427
1000	0.26	0.056	1,735	424
3000	1.21	0.088	1,873	227
6000	4.38	0.111	3,178	121

Table 1.10: Simple communication strategy one to one for *NQP*

Algorithm 17: Sender solver for *NQP* (simple communication strategy)

abstract solver *as_sender* // ITR → number of iterations

computation *: I, V, S, A* // SCI → number of iterations with the same cost

begin

$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ V \ (\mapsto) \ S \ (\mapsto) \ (\!|A|\!)^d \ ]$

end

solver SOLVER$_{sender}$ **implements** as_sender

computation : $I_{perm}, V_{PAS}(p), S_{first}, A_{AI}$

Algorithm 18: Receiver solver for *NQP* (simple communication strategy)

abstract solver *as_receiver* // ITR → number of iterations

computation *: I, V, S, A* // SCI → number of iterations with the same cost

communication *: C.M.*

begin

$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ V \ (\mapsto) \ S \ (\mapsto) \ \left[A \ (?)_{\text{ITR}\%K_2} \ \left[A \ (m) \ C.M.\right]\right] \ ]$

end

solver SOLVER$_{receiver}$ **implements** as_receiver

computation : $I_{perm}, V_{PAS}(p), S_{first}, A_{AI}$

communication: CM_{last}

Algorithm 19: Simple communication strategy one to one for *NQP*

$[\text{SOLVER}_{sender} \cdot A] \ \boxed{\rightarrow} \ [\text{SOLVER}_{receiver} \cdot C.M.] \ 20;$

Algorithm 20: Simple communication strategy one to N for *NQP*

$\left[\text{SOLVER}_{sender} \cdot A(\frac{20}{K})\right] \ \boxed{\rightsquigarrow} \ \left[\text{SOLVER}_{receiver} \cdot C.M.(\frac{20}{K})\right] \ K;$

Tables 1.10 and 1.11 show how the communication improve the non communicating results in terms of runtime and iterations, but this improvement is not significant. In contrast to *SGP*, POSL does not get trapped so often into local minima during the resolution of *NQP*. For that reason, the shared information, once received and accepted by the receivers solvers, does not improves largely the current cost.

***

Instance	Communication 1-n				Communication (1-n)×2				Communication (1-n)×4			
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
250	0.16	0.032	2,621	894	0.15	0.036	2,459	892	0.15	0.036	2,494	547
500	0.20	0.040	1,592	428	0.19	0.053	1,521	539	0.18	0.057	1,719	593
1000	0.26	0.055	1,329	286	0.25	0.046	1,435	369	0.23	0. 056	1,400	426
3000	1.26	0.078	1,657	212	1.22	0.101	1,598	249	1.20	0.078	1,704	252
6000	4.40	0.118	2,771	197	4.35	0.127	2,840	148	4.33	0.120	2,975	188

Table 1.11: Simple communication strategy one to N for *NQP*

In the following experiment, with the goal of improving results, another communication strategy was implemented, very similar to the one applied to *SGP*, but in this case, with solvers using the same neighborhood function $V_{PAS}(p)$ but with different values of p and different selection functions. In this communication strategy a cyclic exchange of the current configuration is performed between to different solvers. One solver *companion* using the neighborhood computation module $V_{PAS}(p)$ with a smaller value of p and using the selection computation module S_{best}, meaning that it is able to find promising configuration faster, but its convergence is slower. The other solver is very similar to the one used for non communicating experiments, but in this communication strategy solvers are both senders and receivers (see Algorithm 21). Before designing communication strategies (Algorithms 22, and 23), many experiments were launched to select: 1. The percentage of variables that the companion solver swaps with the culprit one, when executing the neighborhood computation module (p). This value was decided to be 1. 2. The number of companion solvers to connect with the standard one, for the communication strategy using operator one to N. This vale was decide to be 2, as we can see in Algorithm 23.

Algorithm 21: Solvers for cyclic communication strategy to solve *NQP*

abstract solver *as_cyc* // ITR → number of iterations
computation *:* I, V, S_1, S_2, A // SCI → number of iterations with the same cost
communication *:* $C.M.$
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ V \ (\mapsto) \ S \ (\mapsto) \ \left[A \ (?)_{\text{ITR}\%K_2} \ \left[(\!|A|\!)^d \ (m) \ C.M.\right]\right] \ ]$
end
solver SOLVER$_{standard}$ **implements** as_cyc
computation : $I_{perm}, V_{PAS}(2.5), S_{first}, A_{AI}$
communication: CM_{last}
solver SOLVER$_{companion}$ **implements** as_cyc
computation : $I_{perm}, V_{PAS}(1), S_{best}, A_{AI}$
communication: CM_{last}

Algorithm 22: Cyclic communication strategy one to one for *NQP*

$[\text{SOLVER}_{companion} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{standard} \cdot C.M.] \ 20;$
$[\text{SOLVER}_{standard} \cdot A] \boxed{\rightarrow} [\text{SOLVER}_{companion} \cdot C.M.] \ 20;$

With this experiment, it was possible to find a communication strategy which improves

Algorithm 23: Cyclic communication strategy one to N for *NQP*

$[\text{SOLVER}_{companion} \cdot A(2)]$ $\boxed{\rightsquigarrow}$ $[\text{SOLVER}_{standard} \cdot C.M.]$ 13;
$[\text{SOLVER}_{standard} \cdot A]$ $\boxed{\rightsquigarrow}$ $[\text{SOLVER}_{companion} \cdot C.M.(2)]$ 13;

Instance	Communication 1-1				Communication 1-n				I.R.
	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)	
250	**0.09**	0.021	1,169	254	0.10	0.021	1,224	254	2.00
500	**0.14**	0.027	864	121	0.15	0.030	977	220	1.65
1000	0.22	0.041	889	247	**0.21**	0.056	807	196	1.39
3000	1.25	0.090	1,602	90	**1.02**	0.145	1,613	206	1.17
6000	4.83	0.121	2,938	746	**4.24**	0.746	2,537	779	1.01

Table 1.12: Cyclic communication strategy for *NQP*

runtimes significantly, but only for small instances of the problem, where the number of solutions, with respect to the order N, is lower. This result confirms experimentally the hypothesis already introduced, which propose that as the size of the problem grows, (and with it, the number ef solutions inside the search space with respect to N) lower is the gain using communication during the search process. Table 1.12 shows how the *improvement ratio* (column **I.R.**) decreases with the instance order N. This ratio was computed using the following equation:

$$\frac{\text{runtime without communication}}{\frac{(\text{runtime using communication 1-1}+\text{runtime using communication 1-n})}{2}}$$

1.4 A simple communication strategy (*Costas Array*)

In this section I present the performed study using *Costas Array Problem* (*CAP*) as a benchmark. This time, a simple communication strategy, in which the information to communicate between solvers is the current configuration was tested, showing good results.

1.4.1 Problem definition

The *Costas Array Problem* (*CAP*) consists in finding a *costas array*, which is an $n \times n$ grid containing n marks such that there is exactly one mark per row and per column and the $n(n-1)/2$ vectors joining each couple of marks are all different. This is a very complex problem that finds useful application in some fields like sonar and radar engineering. It also

presents an interesting characteristic: although the search space grows factorially, from order 17 the number of solutions drastically decreases [11].

The cost function for this benchmark was implemented in C++ based on the current implementation of *Adaptive Search*[iii].

1.4.2 Experiment design and results

To handle this problem, I have reused all modules used for solving the *N-Queens Problem*. First attempts to solve this problems were using the same strategies (abstract solvers) used to solve the *Social Golfers Problem* and *N-Queens Problem*, without success: POSL was not able to solve instances larger than $n = 8$ in a reasonable amount of time (seconds). After many unsuccessful attempts to find the right parameters of *maximum number of restarts*, *maximum number of iterations*, and *maximum number of iterations with the same cost*, I decided to implement the mechanism used by Daniel Díaz in the current implementation of *Adaptive Search* to escape from local minima: I have added a *Reset* computation module R_{AS} based on the abstract computation module R. The rest of computation modules were the same used for solving the *N-Queens Problem*.

The basic solver used to solve this problem is presented in Algorithm 24, and it was taken as a base to build all the different communication strategies. Basically, it is a classical local search iteration, where instead of performing restarts, it performs resets. After a deep analysis of this implementation and results of some runs, I decided to use $K_1 = 2,000,000$ (maximum number of iterations) big enough to solve the chosen instance $n = 19$; and $K_2 = 3$ (the number of iteration before performing the next *reset*).

Algorithm 24: Reset-based abstract solver for *CAP*

```
abstract solver as_hard                                  // ITR → number of iterations
computation : I, R, V, S, A
begin
    I (↦) [(↺) (ITR < K1)  R (↦) [(↺) (ITR % K2) [V (↦) S (↦) A] ] ]
end
solver SOLVER_single implements as_hard
    computation : I_perm, R_AS, V_AS, S_first, A_AI
```

Table 1.13 shows results of launching solver sets to solve each instance of *Costas Array Problem* 19 sequentially and in parallel without communication. Runtimes and iteration means showed in this confirm once again the success of the parallel approach.

[iii] It is based on the code from Daniel Díaz available at https://sourceforge.net/projects/adaptivesearch/

STRATEGY	T	T(ds)	It.	It.(sd)	% success
Sequential (1 core)	132.73	80.32	2,332,088	1,424,757	40.00
Parallel (40 cores)	25.51	15.75	231,262	143,789	100.00

Table 1.13: *Costas Array* 19: no communication

***

In order to improve results, a simple communication strategy was applied: communicating the current configuration to other solvers. To do so, we insert a *sending output* operator to the abstract solver in Algorithm 24. This results in the sender solver presented in Algorithm 25.

Algorithm 25: Sender solver for *CAP*

abstract solver *as_hard_sender*
computation : I, R, V, S, A
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ \ T \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} \ \mathbf{\%} \ K_2) \ \left[V \ (\mapsto) \ S \ (\mapsto) \ (\!|A|\!)^d\right] \ ] \ ]$
end
solver SOLVER$_{sender}$ **implements** as_hard_sender
computation : $I_{perm}, R_{AS}, V_{AS}, S_{first}, A_{AI}$

Studying some runs of POSL solving *CAP*, it was observed that the cost of the current configuration of the first solver finding a solution, describes an oscillatory descent due to the repeated resets, but not so pronounced. For that reason, it was decided to apply a simple communication strategy that shares the current configuration while applying the acceptance criterion: its goal is to accelerate the cost descent. To do so, a communication module using a *minimum* operator (m) together with the abstract computation module A was inserted, as shown in Algorithm 26.

One of the main purpose of this study is to explore different communication strategies. We have then implemented and tested different variations of the strategy exposed above by combining two communication operators (one to one and one to N) and different percentages of communicating solvers. For this problem, it was study also the behavior of the communication performed at two different moments: while applying the acceptance criteria (Algorithm 26),

	100% COMM				50% COMM			
STRATEGY	T	T(sd)	It.	It.(sd)	T	T(sd)	It.	It.(sd)
Str A: 1 to 1	11.60	9.17	84,159	68,958	16.78	13.43	148,222	121,688
Str A: 1 to N	**10.83**	8.72	79,551	63,785	13.03	13.46	106,826	120,894
Str B: 1 to 1	14.84	13.54	119,635	112,085	14.51	13.88	125,982	123,261
Str B: 1 to N	22.99	23.82	199,930	207,851	16.62	15.16	138,840	116,858

Table 1.14: *Costas Array* 19: with communication

and while performing a *reset* (Algorithm 27).

Algorithm 26: Receiver solver for *CAP* (variant A)

abstract solver *as_hard_receiver_a* // ITR → number of iterations
computation *: I, T, V, S, A*
communication *: C.M.*
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ \ T \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR } \% \ K_2) \ \left[V \ (\mapsto) \ S \ (\mapsto) \ \left[A \ (m) \ C.M.\right]\right] \ ] \ ]$
end
solver SOLVER$_{receiverA}$ **implements** as_hard_receiver_a
computation : $I_{perm}, R_{AS}, V_{AS}, S_{first}, A_{AI}$
communication: CM_{last}

Algorithm 27: Receiver solver for *CAP* (variant B)

abstract solver *as_hard_receiver_b* // ITR → number of iterations
computation *: I, R, V, S, A*
communication *: C.M.*
begin
$I \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR} < K_1) \ \left[R \ (m) \ C.M.\right] \ (\mapsto) \ [(\circlearrowleft) \ (\text{ITR } \% \ K_2) \ \left[V \ (\mapsto) \ S \ (\mapsto) \ A\right] \ ] \ ]$
end
solver SOLVER$_{receiverB}$ **implements** as_hard_receiver_b
computation : $I_{perm}, R_{AS}, V_{AS}, S_{first}, A_{AI}$
connection: CM_{last}

The instantiation for receiver solvers instantiates the abstract communication module $C.M.$ with the concrete communication module CM_{last}, which takes into account the last received configuration at the time of its execution.

Table 1.14 shows that solver sets executing the strategy A (receiving the configuration at the time of applying the acceptance criteria) is more effective. The reason is that the others, interfere with the proper performance of the *reset*, which is a very important step in the algorithm. This step can be performed on three different ways:

1. Trying to shift left/right all sub-vectors starting or ending by the variable which contributes the most to the cost, and selecting the configuration with the lowest cost.

2. Trying to add a constant (circularly) to each element in the configuration.

3. Trying to shift left from the beginning to some culprit variable (i.e., a variable contributing to the cost).

Then, one of these 3 generated configuration has the same probability of being selected, to be the result of the *reset* step. In that sense, some different *resets* can be performed for the same configuration. Here is when the communication play its important role: receiver and sender solvers apply different *reset* in the same configuration, providing an exploratory factor. Results showed the efficacy of this communication strategy.

Analyzing the whole information obtained during the experiments, we can observe that the percentage of communicating solvers finding the solution thanks to the received information was high. That shows that the communicated information was very helpful during the search process. With the simplicity of the operator-based language provided by POSL, we were able to find a simple communication strategy to obtain better results than applying sequential and parallel independent multi-walk approaches. As expected, the best strategy was based on 100% of communication and a one to N communication, because this strategy allows to communicate a promising place inside the search space to a maximum of solvers, helping the decisive intensification process. Algorithm 28 shows the code of this communication strategy. Using the one to N operator $\boxed{\rightsquigarrow}$ each sender solver sends information to every receiver solver.

Algorithm 28: Communication strategy one to N 100% for *CAP*

$[\text{SOLVER}_{sender} \cdot A(20)] \boxed{\rightsquigarrow} [\text{SOLVER}_{receiverA} \cdot C.M.(20)]$;

Table 1.14 shows also high values of standard deviation. This is not surprising, due to the highly random nature of the neighborhood function and the selecting criterion, as well as the execution of many resets during the search process.

1.5 A local minima evasion strategy (*Golomb Ruler*)

In this section, the performed study using *Golomb Ruler Problem* (*GRP*) as a benchmark is presented. Using this benchmark, a different communication strategy was tested: we communicate the current configuration in order to avoid its neighborhood, i.e., a *tabu* configuration.

1.5.1 Problem definition

The *Golomb Ruler Problem* (*GRP*) problem consists in finding an ordered vector of n distinct non-negative integers, called *marks*, $m_1 < \cdots < m_n$, such that all differences $m_i - m_j$ $(i > j)$ are all different. An instance of this problem is the pair (o, l) where o is the order of the problem, (i.e., the number of *marks*) and l is the length of the ruler (i.e., the last *mark*). We assume that the first *mark* is always 0. This problem has been applied to radio astronomy, x-ray crystallography, circuit layout and geographical mapping [12]. When POSL is applied to solve an instance of this problem sequentially, It can be noticed that it performs many *restarts* before finding a solution. For that reason this problem was chosen to study a new communication strategy.

The cost function is implemented based on the storage of a counter for each measure present in the rule (configuration). All distances where a variable is participating are also stored. This information is useful to compute the more culprit variable (the variable that interferes less in the represented measures), in case of the user wants to apply algorithms like *Adaptive Search*. This cost is calculated in $O\left(o^2 + l\right)$.

1.5.2 Experiment design and results

Golomb Ruler Problem instances were used to study a different communication strategy. This time the current configuration is communicated, to avoid its neighborhood, i.e., a *tabu* configuration. Some modules used in the resolution of *Social Golfers* and *Costas Array* problems have been reused to design the solvers: the *Selection* and *Acceptance* modules. The new modules are:

1. I_{sort}: returns a random configuration s as an ordered vector of integers. The configuration is generated *far* from the set of *tabu* configurations arrived via solver-communication (in communicating strategies) (based on the *generation* abstract module I).

2. V_{sort}: given a configuration, returns the neighborhood $V\left(s\right)$ by changing one value while keeping the order, i.e., replacing the value s_i by all possible values $s'_i \in D_i$ satisfying $s_{i-1} < s'_i < s_{i+1}$ (based on the *neighborhood* abstract module V).

It was also added an abstract module R for reset: it receives and returns a configuration. The concrete reset module used for this problem (R_{tabu}) inserts the input configuration into a *tabu* list inside the solver and returns the input configuration as-is. As Algorithm 30 shows, this module is executed just before performing a restart, so the solver was unable to find a

better configuration around the current one. Therefore, the current configuration is assumed to be a local minimum, and it is inserted into a tabu list.

Algorithm 29 and 30 present both solvers, using a tabu list and without using it. They were used to obtain results presented in Tables 1.15 and 1.16 to show that the approach explained above provides some gain in terms of runtime.

Algorithm 29: Solver without using tabu list, for *GRP*

```
abstract solver as_golomb_notabu                    // Itr → number of iterations
computation : I, V, S, A
begin
    [(↺) (Itr < K1)  I (↦)  [(↺) (Itr % K2) [V (↦) S (↦) A] ]  ]
end
solver Solver_notabu implements as_notabu
    computation : I_sort, V_sort, S_first, A_AI
```

Algorithm 30: Solver using tabu list, for *GRP*

```
abstract solver as_golomb_tabu                      // Itr → number of iterations
computation : I, V, S, A
begin
    [(↺) (Itr < K1)  I (↦)  [(↺) (Itr % K2) [V (↦) S (↦) A] ]  (↦) (|T|)^o  ]
end
solver Solver_tabu implements as_tabu
    computation : I_sort, V_sort, S_first, A_AI
```

Instance	T	T(sd)	It.	It.(sd)	R	R(sd)	% success
8–34	0.79	0.66	13,306	11,154	66	55.74	100.00
10–55	66.44	49.56	451,419	336,858	301	224.56	80.00
11–72	160.34	96.11	431,623	272,910	143	90.91	26.67

Table 1.15: A single sequential solver without using tabu list for *GRP*

Instance	T	T(sd)	It.	It.(sd)	R	R(sd)	% success
8–34	0.66	0.63	10,745	10,259	53	51.35	100.00
10–55	67.89	50.02	446,913	328,912	297	219.30	88.00
11–72	117.49	85.62	382,617	275,747	127	91.85	30.00

Table 1.16: A single sequential solver using tabu list for *GRP*

The benefit of the parallel approach is also proved for the *Golomb Ruler Problem*, as we can see in Table 1.17. However, without communication, the improvement is not substantial (8% for 8–34, 7% for 10–55 and 5% for 11–72). The reason is because only one configuration is inserted in the tabu list after each restart.

***

Instance	T	T(sd)	It.	It.(sd)	R	R(sd)
8–34	0.43	0.37	349	334	1	1.64
10–55	4.92	4.68	20,504	19,742	13	13.07
11–72	85.02	67.22	155,251	121,928	51	40.64

Table 1.17: Parallel solvers using tabu list for *GRP*

The main goal of choosing this benchmark was to study a different communication strategy, since for solving this problem, POSL needs to perform some restarts. In this communication strategy, solvers do not communicate the current configuration to have more solvers searching in its neighborhood, but a configuration that we assume is a local minimum to be avoided. We consider that the current configuration is a local minimum since the solver (after a given number of iteration) is not able to find a better configuration in its neighborhood, so it will communicate this configuration just before performing the restart.

Algorithm 31 presents the sender solver and Algorithm 32 presents the receiver solver. Based on the connection operator used in the communication strategy, this solver might receives one or many configurations. These configurations are the input of the generation module (I), and this module inserts all received configurations into a *tabu* list, and then generates a new first configuration, far from all configurations in the *tabu* list.

Algorithm 31: Sender solver for *GRP*

```
abstract solver as_golomb_sender                    // Itr → number of iterations
computation : I, V, S, A, R
begin
    [↻ (Itr < K1)  I ↦ [↻ (Itr % K2) [V ↦ S ↦ A] ] ↦ (|R|)^o ]
end
solver Solver_sender implements as_golomb_sender
    computation : I_sort, V_sort, S_first, A_AI, R_tabu
```

Algorithm 32: Receiver solver for *GRP*

```
abstract solver as_golomb_receiver                  // Itr → number of iterations
computation : I, V, S, A, R
connection : C.M.
begin
    [↻ (Itr < K1)  [C.M. ↦ I] ↦ [↻ (Itr % K2) [V ↦ S ↦ A] ] ↦ R ]
end
solver Solver_receiver implements as_golomb_receiver
    computation : I_sort, V_sort, S_first, A_AI, R_tabu
    communication: CM_last
```

In this communication strategy there are some parameters to be tuned. The first ones are: 1. K_1, the number of restarts, and 2. K_2, the number of iterations by restart. Both are instance dependent, so, after many experimental runs, I choose them as follows:

- *Golomb Ruler* 8–34: $K_1 = 300$ and $K_2 = 200$

- *Golomb Ruler* 10–55: $K_1 = 1000$ and $K_2 = 1500$
- *Golomb Ruler* 11-72: $K_1 = 1000$ and $K_2 = 3000$

The idea of this strategy (abstract solver) follows the following steps:

Step 1

The computation module I_{sort} generates an initial configuration tacking into account a set of configurations into a tabu list. The configuration arriving to this tabu list come from the same solver (Step 3) and/or from outside (other solvers) depending on the strategy (non-communicating or communicating), and on the type of the solver (sender or receiver).

This module executes some other modules provided by POSL to solve the *Sub-Sum Problem* in order to generates *valid* configurations for *Golomb Ruler Problem*. A valid configuration s for *Golomb Ruler Problem* is a configuration that fulfills the following constraints:

- $s = (a_1, \ldots, a_o)$ where $a_i < a_j, \forall i < j$, and
- all $d_i = a_{i+1} - a_i$ are all different, for all $d_i, i \in [1...o-1]$

The *Sub-sum Problem* is defined as follows: Given a set E of integers, with $|E| = N$, finding a sub set e of n elements that sums exactly z. In that sense, a solution $S_{sub-sum} = \{s_1, \ldots, s_{o-1}\}$ of the *Sub-sum problem* with $E = \left\{1, \ldots, l - \frac{(o-2)(o-1)}{2}\right\}$, $n = o - 1$ and $z = l$, can be traduced to a *valid* configuration C_{grp} for *Golomb Ruler Problem* as follows:

$$C_{grp} = \{c_1, c_1 + s_1, \ldots, c_{o-1} + s_{o-1}\}$$

where $c_1 = 0$.

In the selection module applied inside the module I, the selection step of the search process selects a configuration from the neighborhood *far* from the *tabu* configurations, with respect to certain vectorial norm and an epsilon. In other words, a configuration C is selected if and only if:

1. the cost of the configuration C is lower than the current cost, and
2. $||C - C_t||_p > \varepsilon$, for all *tabu* configuration C_t

where p and ε are parameters.

I experimented with 3 different values for p. Each value defines a different type of norm of a vector $x = \{x_1, \ldots, x_n\}$:

- $p = 1$: $||x||_1 = \sum_{i=0}^{n} |x_i|$
- $p = 2$: $||x||_2 = \sqrt{\sum_{i=0}^{n} |x_i|^2}$
- $p = \infty$: $||x||_\infty = \max(x)$

After many experimental runs with these values I choose $p = \infty$ and $\varepsilon = 4$ for the study of the communication strategy. I also made experiments trying to find the right size for the $tabu$ list and the conclusion was that the right sizes were 15 for non-communicating strategies and 40 for communicating strategies, taking into account that in the latter, I work with 20 receivers solvers.

Step 2

After generating the first configuration, the next step is to apply a local search to improve it. In this step I use the neighborhood computation module V_{sort}, that creates neighborhood $\mathcal{V}(s)$ by changing one value while keeping the order in the configuration, and the other modules (selection and acceptance). The local search is executed a number K_2 of times, or until a solution is obtained.

Step 3

If no improvement is reached, the current configuration is classified as a *potential local minimum* and inserted into the $tabu$ list, then send it (on the case of sender solvers). Then, the process returns to the Step 1.

The POSL code of the communication strategy using the one to N operator is presented in Algorithm 33.

Algorithm 33: Communication strategy one to N for *GRP*

$[\text{SOLVER}_{sender} \cdot R(20)] \boxed{\rightsquigarrow} [\text{SOLVER}_{receiver} \cdot C.M.(20)]$;

When we use communication one to one, after k restarts the receiver solver has $2k$ configurations inside its tabu list: its own tabu configurations and the received ones. Table 1.18 shows that this strategy is not sufficient for some instances, but when we use communication one to N, the number of tabu configurations after k restarts in the receiver solver is considerably higher: $k(N+1)$: its own tabu configurations and the ones received from N sender solvers the receiver solver is connected with. Hence, these solvers can generate configurations far enough from many potentially local minima. This phenomenon is more visible when the problem order o increases. Table 1.19 shows that the improvement for the higher case (11-72) is about 29% w.r.t non communicating solvers (Table 1.17).

Instance	T	T(sd)	It.	It.(sd)	R	R(sd)
8–34	0.44	0.31	309	233	1	1.23
10–55	3.90	3.22	15,437	12,788	10	8.52
11–72	85.43	52.60	156,211	97,329	52	32.43

Table 1.18: *Golomb Ruler*: parallel, communication one to one

Instance	T	T(sd)	It.	It.(sd)	R	R(sd)
8–34	**0.43**	0.29	283	225	1	1.03
10–55	**3.16**	2.82	12,605	11,405	8	7.61
11–72	**60.35**	43.95	110,311	81,295	36	27.06

Table 1.19: *Golomb Ruler*: parallel, communication one to N

1.6 Summarizing

In this chapter various Constraint Satisfaction Problems as benchmarks have been chosen to 1. evaluate the POSL behavior solving these kind of problems, and 2. to study different solution strategies, specially communication strategies. To this end, benchmarks with different characteristics have been selected, to help me having a wide view of the POSL behavior.

In the solution process of *Social Golfers Problem*, it was studied an exploitation-oriented communication strategy, in which the current configuration is communicated to ask other solvers for help to concentrate the effort in a more promising area. Results show that this communication strategy can provide some gain in terms of runtime. It was also presented results showing the success of a cost descending acceleration communication strategy, exchanging the current configuration between two solvers with different characteristics. Some other unsuccessful communication strategies were studied, showing that the sub-division of the effort by weeks, do not work well. Table 1.20 summarizes the obtained results solving *SGP*.

Instance	Sequential		Parallel		Cooperation	
	T	It.	T	It.	T	It.
5–3–7	1.25	2,907	0.23	142	**0.08**	139
8–4–7	0.60	338	0.28	93	**0.14**	100
9–4–8	1.04	346	0.60	139	**0.36**	144

Table 1.20: Summarizing results for *SGP*

It was showed that simple communication strategies as they were applied to solve *Social Golfers Problem* does not improve enough the results without communication for the *N-Queens Problem*. In this problem, the number of solution with respect to the order N increase

exponentially, then higher order instances are "easer" to solve using local search. For that reason, the communication can not provide a lot on gain. However, a deep study of the POSL's behavior during the search process allowed to design a communication strategy able to improve the results obtained using non-communicating strategies for small instances. Table 1.21 summarizes the obtained results solving *NQP*.

Instance	Sequential		Parallel		Cooperation	
	T	It.	T	It.	T	It.
250	0.29	8,898	0.19	4,139	**0.09**	1,169
500	0.35	4,203	0.24	2,675	**0.14**	864
1000	0.35	2,766	0.30	2,102	**0.21**	807
3000	1.50	2,191	1.33	2,168	**1.02**	1,613
6000	4.71	3,339	4.57	3,323	**4.24**	2,537

Table 1.21: Summarizing results for *NQP*

The *Costas Array Problem* is a very complicated constrained problem, and very sensitive to the methods to solve it. For that reason I used some ideas from already existing algorithms. However, thanks to some studies of different communication strategies, based on the communication of the current configuration at different times (places) in the algorithm, it was possible to find a communication strategy to improve the performance. Table 1.22 summarizes the obtained results solving *CAP*.

STRATEGY	T	It.	% success
Sequential	132.73	2,332,088	40.00
Parallel	25.51	231,262	100.00
Cooperative Strategy	**10.83**	79,551	100.00

Table 1.22: Summarizing results for *CAP* 19

During the solution process of the *Golomb Ruler Problem*, POSL needs to perform many restarts. For that reason, this problem was chosen to study a different (and innovative up to my knowledge) communication strategy, in which the communicated information is a potential local minimum to be avoided. This new communication strategy showed to be effective to solve these kind of problems. Table 1.23 summarizes the obtained results solving *GRP*.

Instance	Sequential				Parallel			Cooperation		
	T	It.	R	% success	T	It.	R	T	It.	R
8–34	0.66	10,745	53	100.00	0.43	349	1	**0.43**	283	1
10–55	67.89	446,913	297	88.00	4.92	20,504	13	**3.16**	12,605	8
11–72	117.49	382,617	127	30.00	85.02	155,251	51	**60.35**	110,311	36

Table 1.23: Summarizing results for *GRP*

In all cases, thanks to the operator-based language provided by POSL it was possible to test many different strategies (communicating and non-communicating) fast and easily. Whereas

creating solvers implementing different solution strategies can be complex and tedious, POSL gives the possibility to make communicating and non-communicating solver prototypes and to evaluate them with few efforts. In this chapter it was possible to show that a good selection and management of inter-solvers communication can help to the search process, working with complex constrained problems.

2

Bibliography

[1] Alexander E.I. Brownlee, Jerry Swan, Ender Özcan, and Andrew J. Parkes. Hyperion 2. A toolkit for {meta- , hyper-} heuristic research. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1133–1140, Vancouver, BC, 2014. ACM.

[2] Alex S Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation*, 16(1):31–61, 2008.

[3] Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, and Christophe Ponsard. Combining Neighborhoods into Local Search Strategies. In *11th MetaHeuristics International Conference*, Agadir, 2015. Springer.

[4] Sébastien Cahon, Nordine Melab, and El-Ghazali Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[5] Simon Martin, Djamila Ouelhadj, Patrick Beullens, Ender Ozcan, Angel A Juan, and Edmund K Burke. A Multi-Agent Based Cooperative Approach To Scheduling and Routing. *European Journal of Operational Research*, 2016.

[6] Alejandro Reyes-amaro, Éric Monfroy, and Florian Richoux. POSL: A Parallel-Oriented metaheuristic-based Solver Language. In *Recent developments of metaheuristics*, to appear. Springer.

[7] Frédéric Lardeux, Éric Monfroy, Broderick Crawford, and Ricardo Soto. Set Constraint Model and Automated Encoding into SAT: Application to the Social Golfer Problem. *Annals of Operations Research*, 235(1):423–452, 2014.

[8] Daniel Diaz, Florian Richoux, Philippe Codognet, Yves Caniou, and Salvador Abreu. Constraint-Based Local Search for the Costas Array Problem. In *Learning and Intelligent Optimization*, pages 378–383. Springer, 2012.

[9] Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1–31, 2009.

[10] Rok Sosic and Jun Gu. Effcient Local Search with Conflict Minimization: A Case Study of the N-Queens Problem. *IEEE Transactions on Knowledge and Data Engineering*, 6:661–668, 1994.

[11] Konstantinos Drakakis. A review of Costas arrays. *Journal of Applied Mathematics*, 2006:32 pages, 2006.

[12] Stephen W. Soliday, Abdollah. Homaifar, and Gary L. Lebby. Genetic algorithm approach to the search for Golomb Rulers. In *International Conference on Genetic Algorithms*, volume 1, pages 528–535, Pittsburg, 1995.

Part II

Appendix

A

Results of experiments with *Social Golfers Problem*

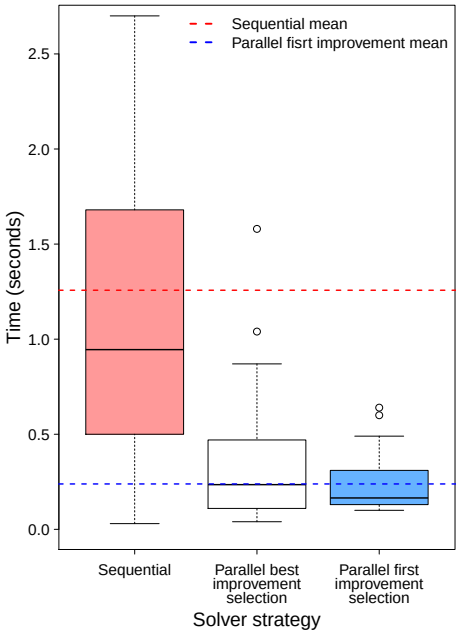
This Appendix presents graphically a summary of individuals runs using Social Golfers Problem. Figures show a box-plot representation for different strategies and a bar representation for the percentage of winner solvers types.

In Figures A.2, A.3 and A.4, labels of the x-axis correspond to the following strategies:

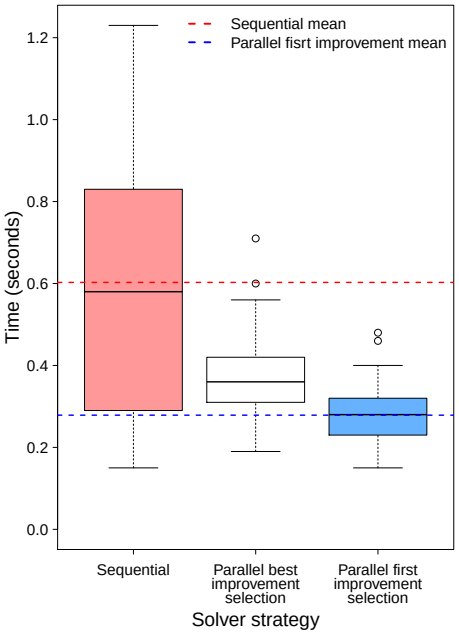
NC: Non communication strategy
100SC1-1: 100% of communicating solvers performing simple communication one to one
50SC1-1: 50% of communicating solvers performing simple communication one to one
25SC1-1: 25% of communicating solvers performing simple communication one to one
100SC1-n: 100% of communicating solvers performing simple communication one to N
50SC1-n: 50% of communicating solvers performing simple communication one to N
25SC1-n: 25% of communicating solvers performing simple communication one to N
CC1-n: One set of solvers performing dynamic exchange communication one to N
CC1-n/2: Two sets of solvers performing dynamic exchange communication one to N
CC1-n/4: Four sets of solvers performing dynamic exchange communication one to N
100CC1-1: 100% of communicating solvers performing dynamic exchange communication one to one
50CC1-1: 50% of communicating solvers performing dynamic exchange communication one to one
25CC1-1: 25% of communicating solvers performing dynamic exchange communication one to one

A.1 Comparison between sequential and parallel runs

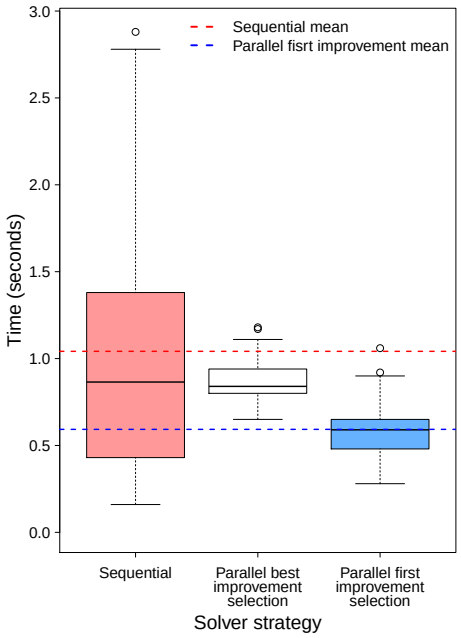
Figure A.1 shows the huge difference between the spread in results of sequential and parallel runs. Although both parallel strategies (best and first improvement) are equally stable, the one using the selection computation module of first improvement shows better results. All of their runs are under the mean value of sequential runs (except a suspected outlier).



(a) *SGP* 5-3-7



(b) *SGP* 8-4-7



(c) *SGP* 9-4-8

Figure A.1: Comparison between sequential and parallel (best improvement and first improvement selections) runs to solve *SGP* using POSL

A.2 Comparison between communication strategies

Figures A.2, A.3 and A.4 show results summaries of different communication strategies. They are presented together with the box-plot graph of parallel results without communication (`NC`). In all cases we can observe that strategies `100%CC1-1` and `50%CC1-1` show excellent results in comparison to the one without communication. This figure shows also that simple communication strategies do not contribute enough to the improvement of search time.

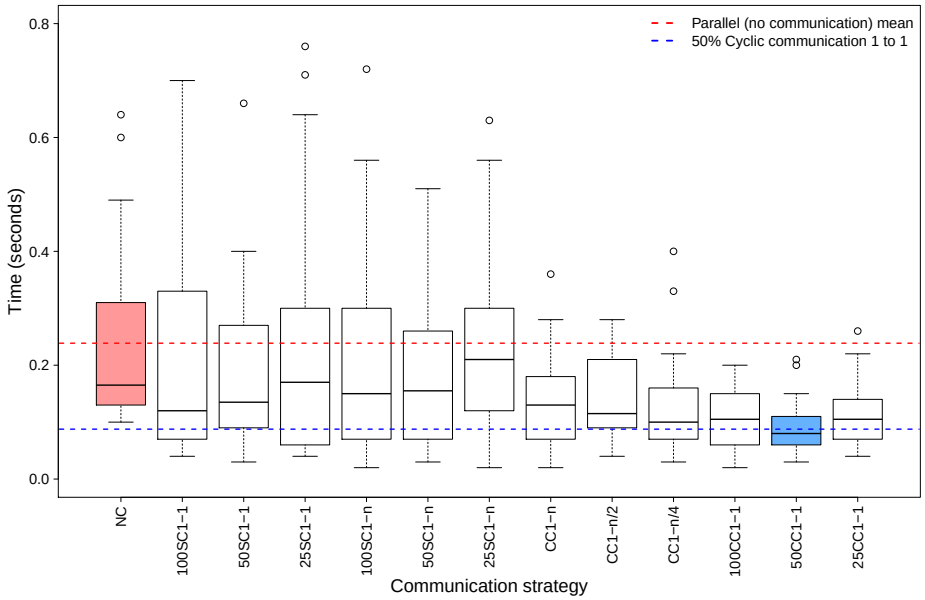


Figure A.2: Different communication strategies to solve *SGP* 5-3-7 using POSL

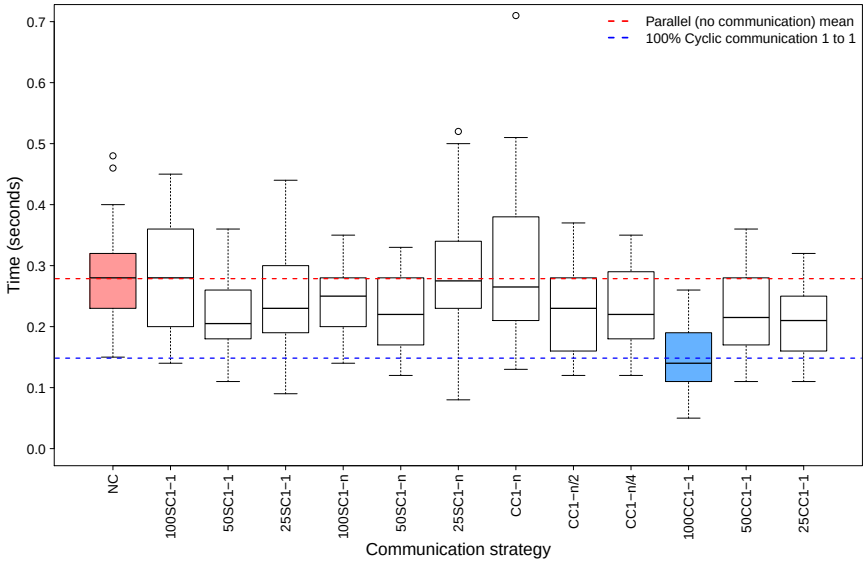


Figure A.3: Different communication strategies to solve *SGP* 8-4-7 using POSL

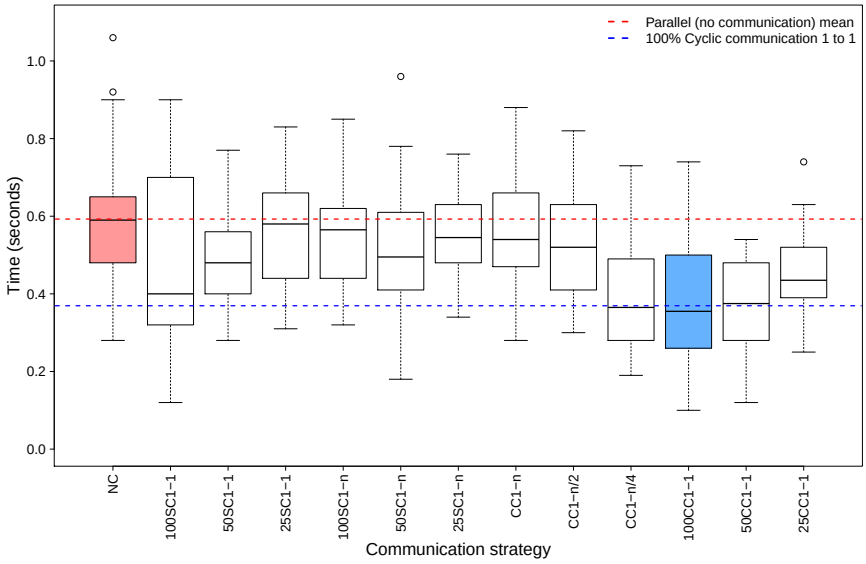


Figure A.4: Different communication strategies to solve *SGP* 9-4-8 using POSL

A.3 Winner solver type representation

Figures A.5, A.6 and A.7, represent the percentage of winner solvers for each communication strategy, according to four different types:

Receiver:	Receiver solver wining thanks to the received information
Sender:	Sender solver
Passive receiver:	Receiver solver wining without using the received information
Non communicating:	Non communicating solver

In these bar graphs is evident to see how the solver communication has played an important role in the solution process. The number of receiver solvers which have win the search process is high, as expected, in dynamic exchange strategies with communication one to N. However, due to the huge traffic of information, the communication overhead make the resulting runtimes not competitive.

The other interesting detail showed in these graphs is that this number of winner receiver solvers remains high in the winner communication strategies (`100%CC1-1` and `50%CC1-1`), achieving an important trade-off between information traffic and search speed.

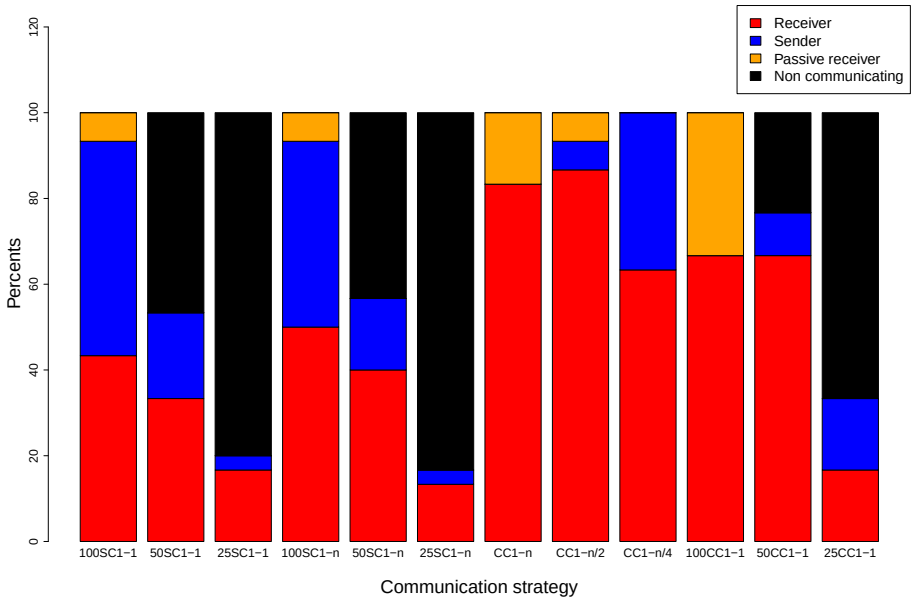


Figure A.5: Solver proportion for each communication strategy to solve *SGP* 5-3-7 using POSL

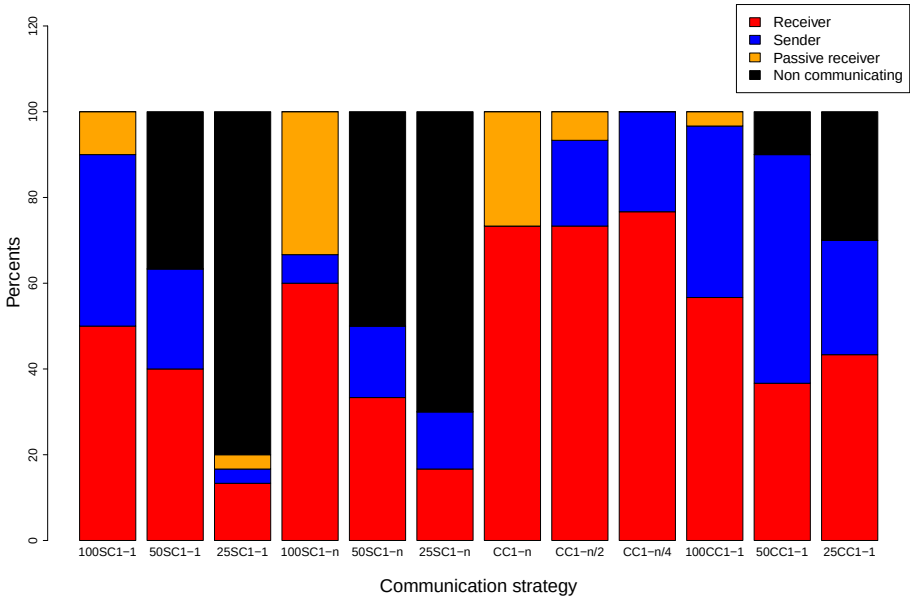


Figure A.6: Solver proportion for each communication strategy to solve *SGP* 8-4-7 using POSL

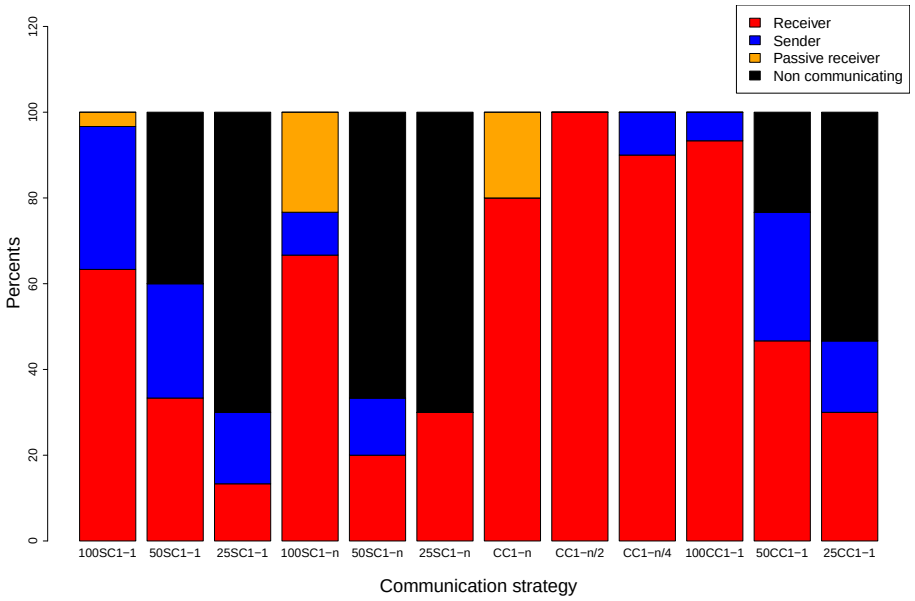


Figure A.7: Solver proportion for each communication strategy to solve *SGP* 9-4-8 using POSL

B

Results of experiments with *N-Queens Problem*

This Appendix, presents graphically a summary of individuals runs using N-Queens Problem. Figures show a box-plot representation for different strategies and a bar representation for the percentage of winner solvers types.

In Figures B.1, B.2, B.3, B.4 and B.5, labels of the x-axis correspond to the following strategies:

Seq: Sequential strategy (1 core)
NC: Parallel non communicative strategy
Cyc1-1: Cyclic communicating strategy with communication one to one
Cyc1-n: Cyclic communicating strategy with communication one to N
S1-1: Simple communicating strategy with communication one to one
S1-n: One set of solvers performing a simple communicating strategy with communication one to one
S1-n/2: Two sets of solvers performing a simple communicating strategy with communication one to one
S1-n/4: Four sets of solvers performing a simple communicating strategy with communication one to one

Figures B.6a, B.6b, B.6c, B.6d and B.7, represent the percentage of winner solvers for each communication strategy, according to four different types:

Receiver: Receiver solver wining thanks to the received information
Sender: Sender solver
Passive receiver: Receiver solver wining without using the received information

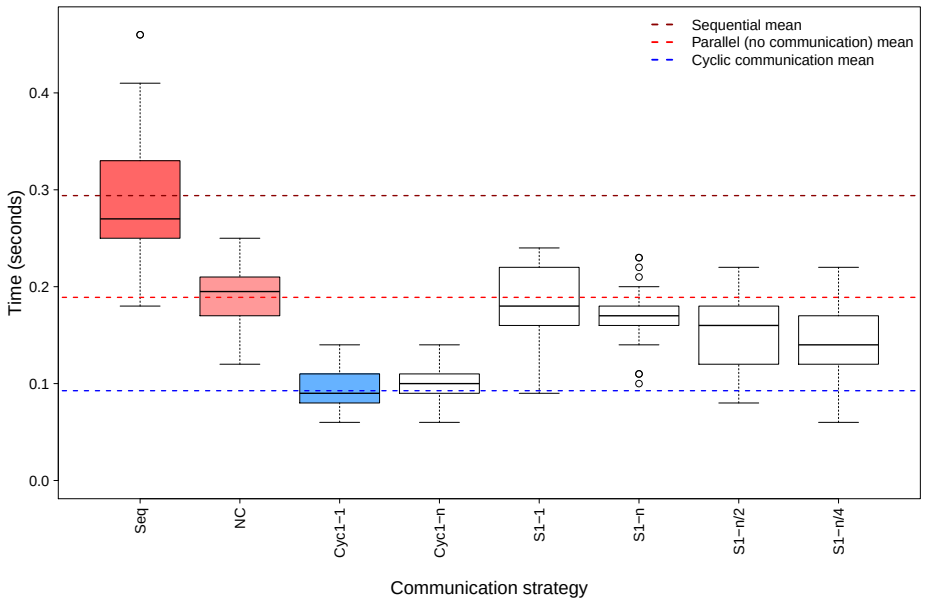


Figure B.1: Different communication strategies to solve 250-Queens using POSL

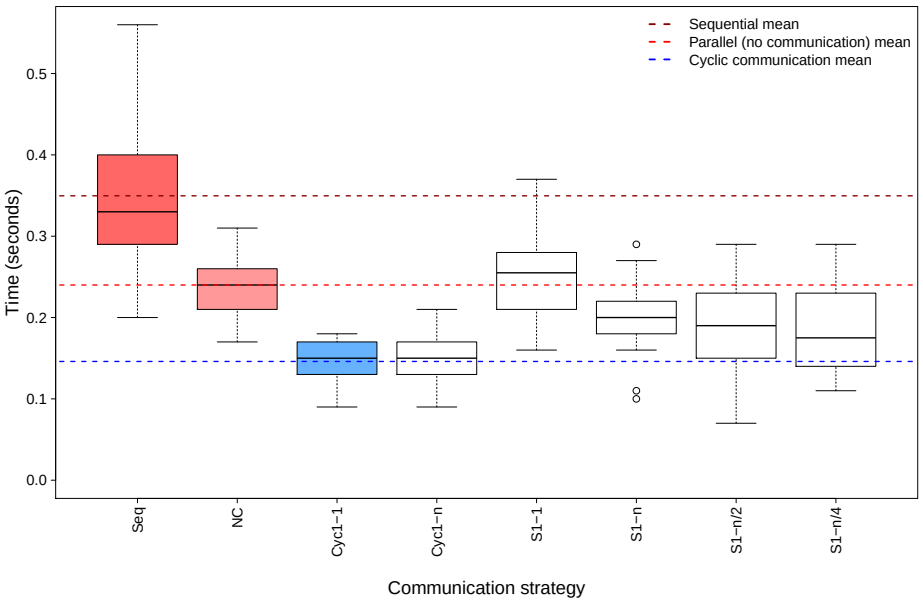


Figure B.2: Different communication strategies to solve 500-Queens using POSL

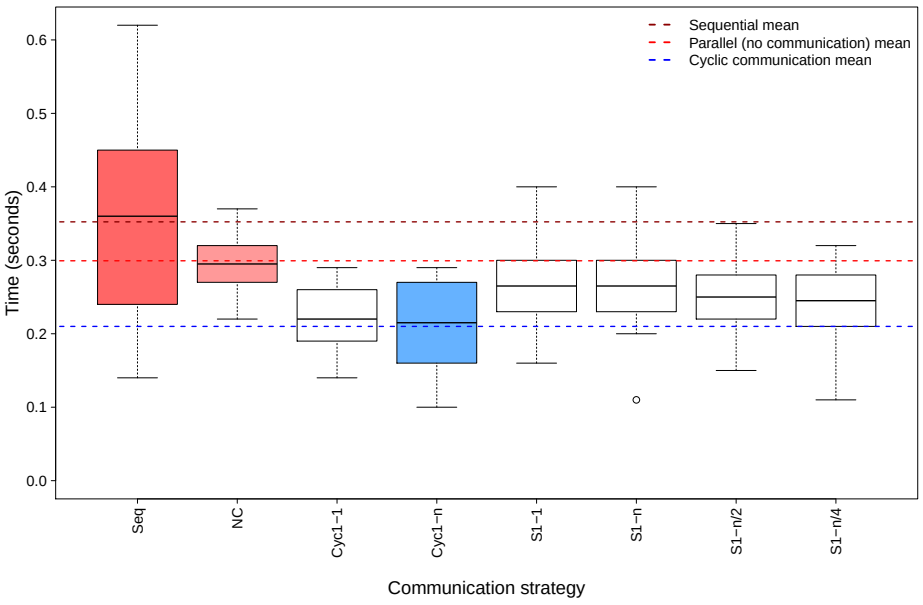


Figure B.3: Different communication strategies to solve 1000-Queens using POSL

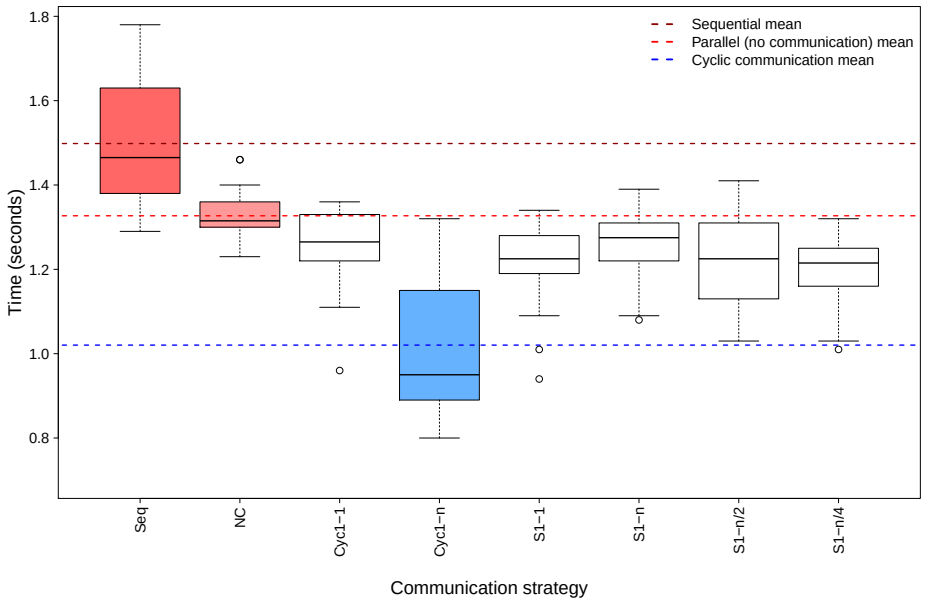


Figure B.4: Different communication strategies to solve 3000-Queens using POSL

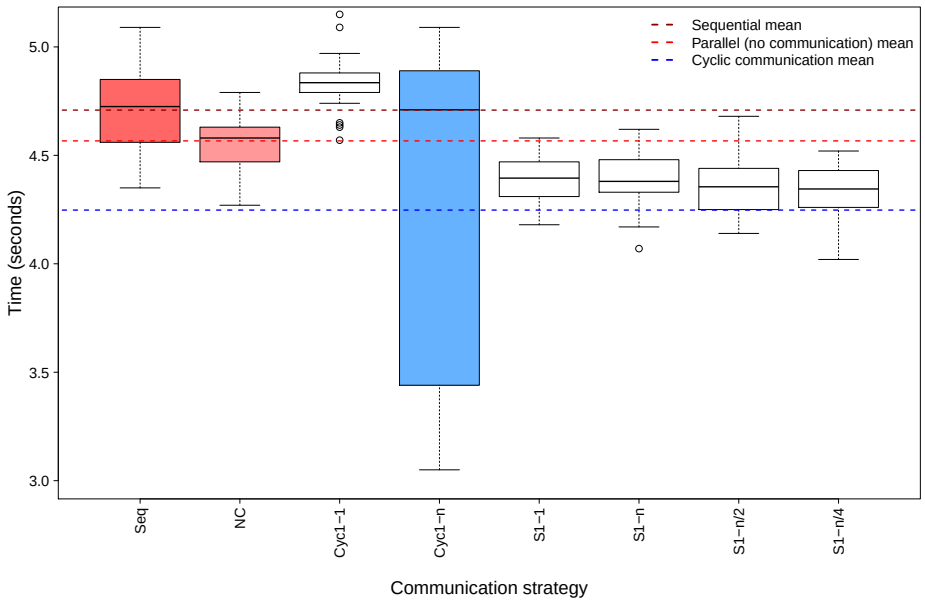
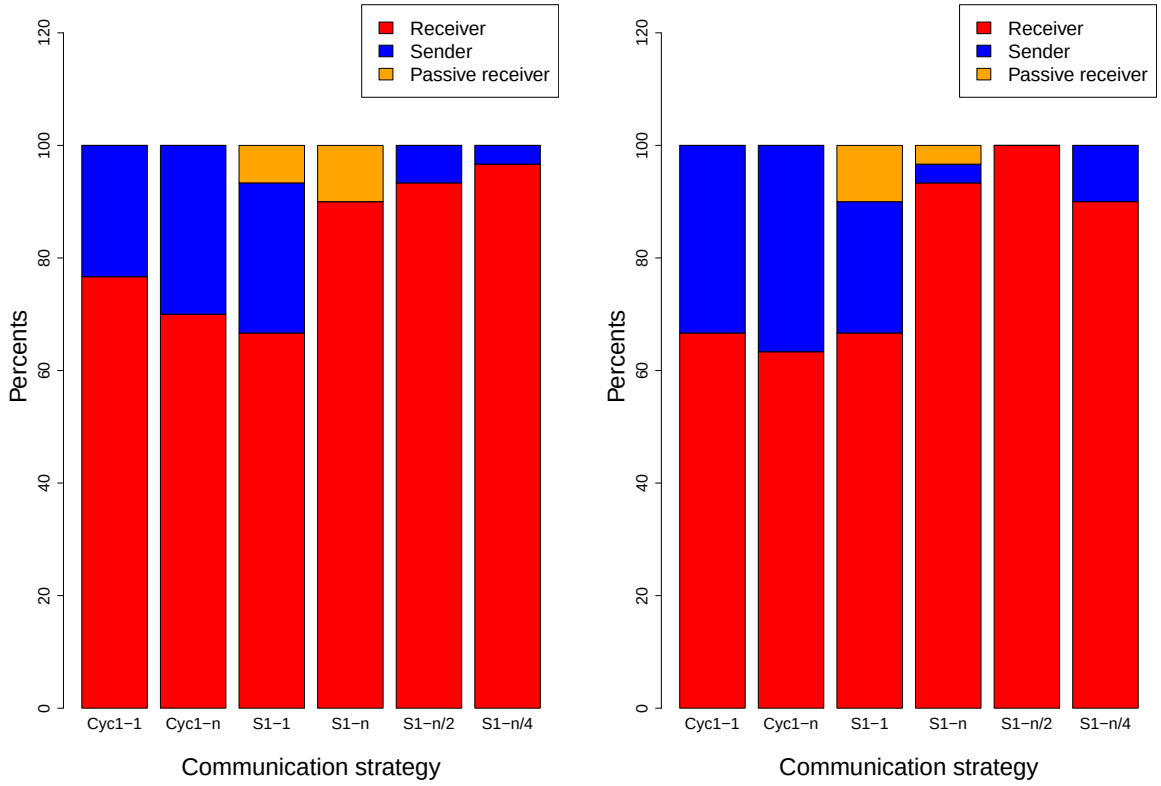
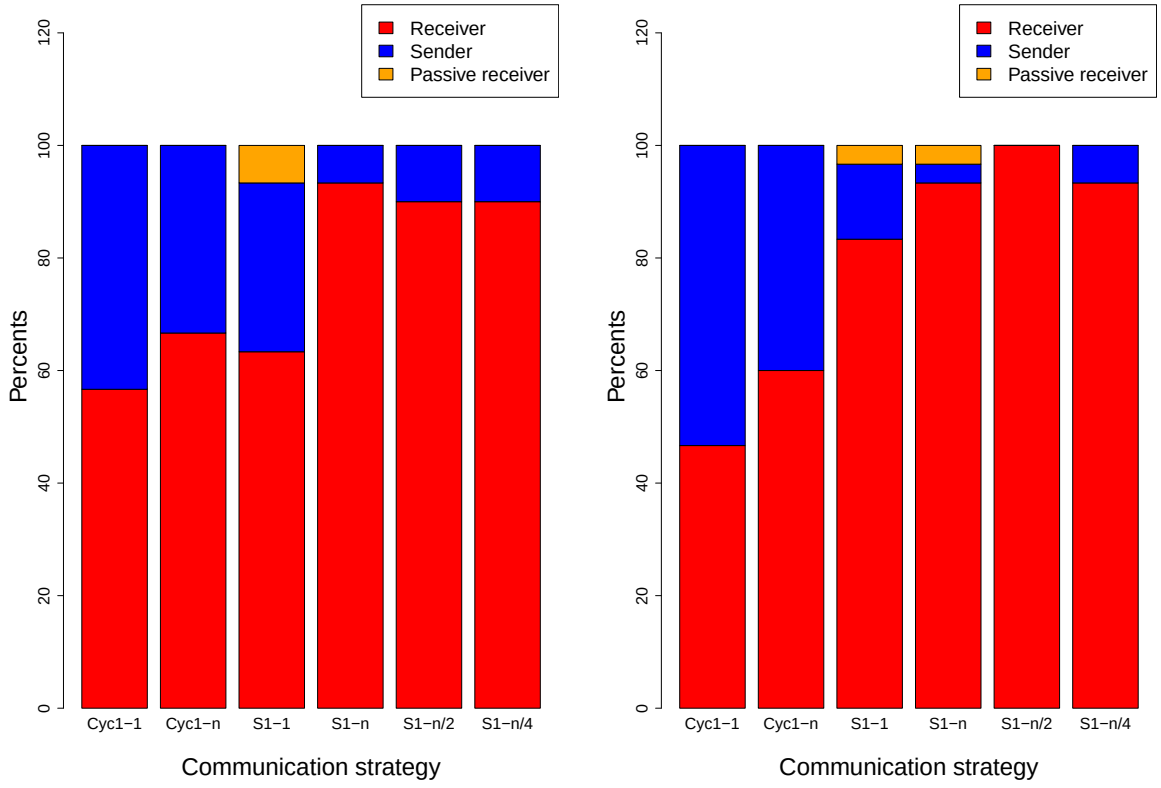


Figure B.5: Different communication strategies to solve 6000-Queens using POSL



(a) NQP 250 **(b)** NQP 500



(c) NQP 1000 **(d)** NQP 3000

Figure B.6: Solver proportion for each communication strategy to solve NQP using POSL

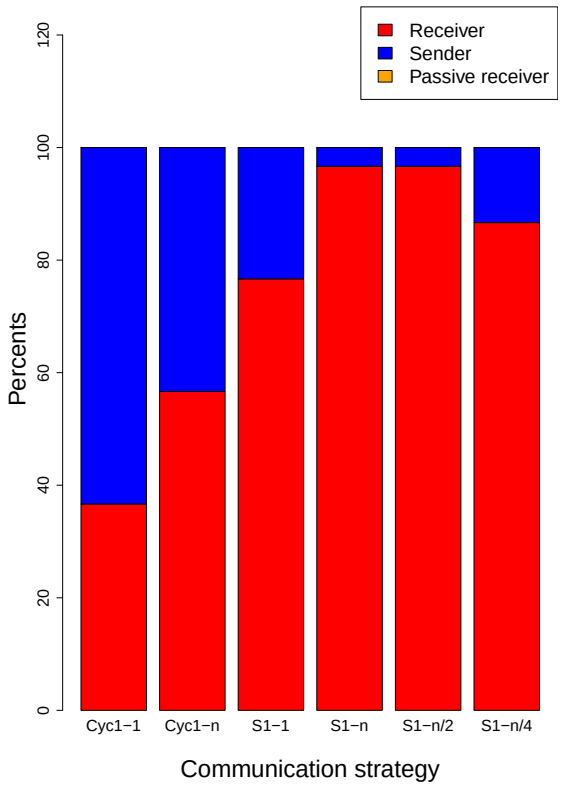


Figure B.7: Solver proportion for each communication strategy to solve *NQP* using POSL

C

Results of experiments with *Costas Array Problem*

This Appendix presents graphically a summary of individuals runs using Costas Array Problem. Figures show a box-plot representation for different strategies and a bar representation for the percentage of winner solvers types.

In Figure C.2, labels of the x-axis correspond to the following strategies:

NC:	Non communication strategy
A1-1:	100% of communicating solvers performing the communication strategy A one to one
B1-1:	100% of communicating solvers performing the communication strategy B one to one
A1-n:	100% of communicating solvers performing the communication strategy A one to N
B1-n:	100% of communicating solvers performing the communication strategy B one to N
50A1-1:	50% of communicating solvers performing the communication strategy A one to one
50B1-1:	50% of communicating solvers performing the communication strategy B one to one
50A1-n:	50% of communicating solvers performing the communication strategy A one to N
50B1-n:	50% of communicating solvers performing the communication strategy B one to N

Figure C.3 represents the percentage of winner solvers for each communication strategy, according to four different types:

Receiver:	Receiver solver wining thanks to the received information
Sender:	Sender solver
Non communicating:	Non communicating solver

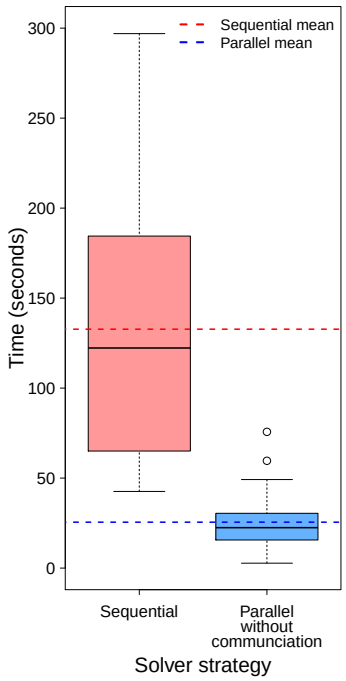


Figure C.1: Comparison between sequential and parallel runs to solve *CAP* 19 using POSL

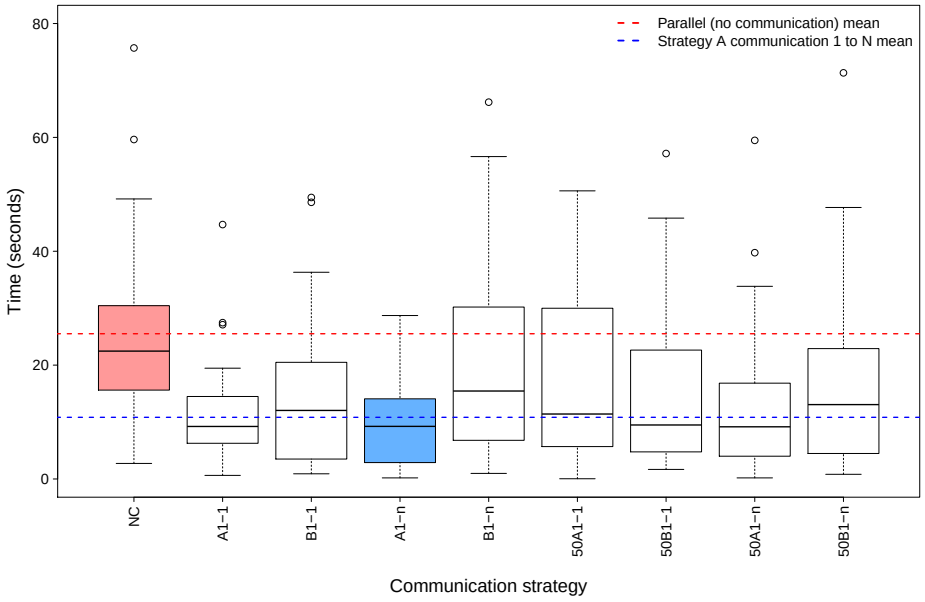


Figure C.2: Different communication strategies to solve *CAP* 19 using POSL

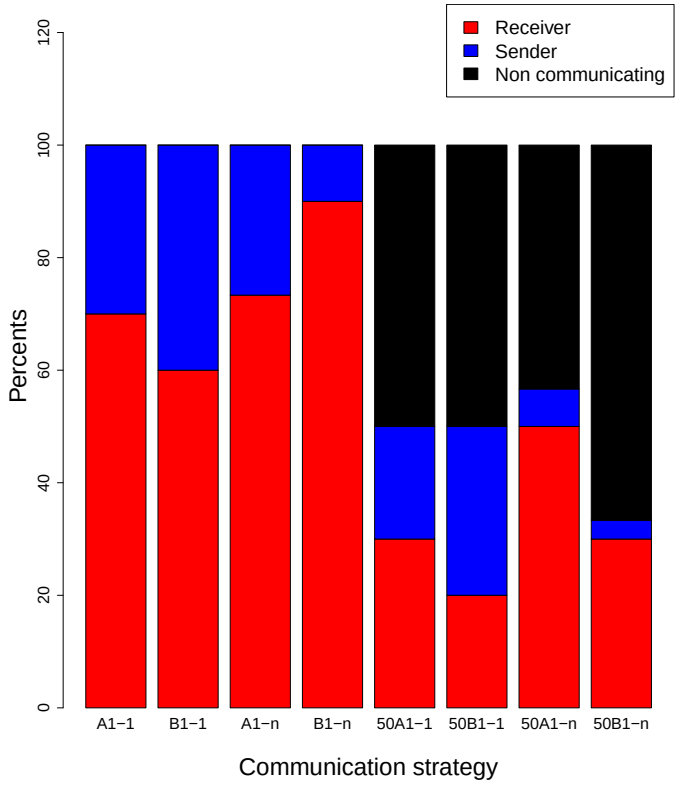


Figure C.3: Solver proportion for each communication strategy to solve *CAP* 19 using POSL

D

Results of experiments with *Golomb Ruler Problem*

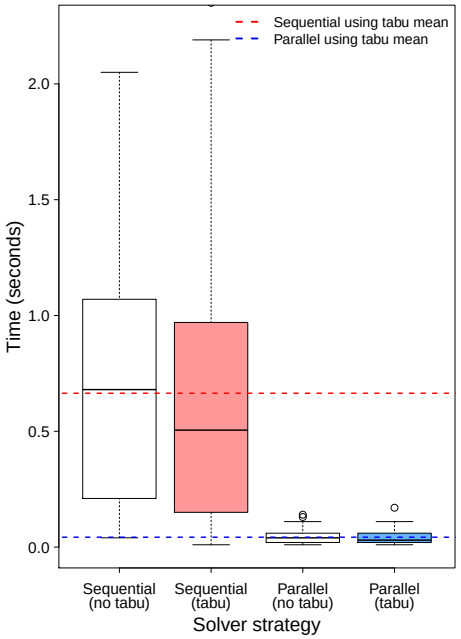
This Appendix, presents graphically a summary of individuals runs using Golomb Ruler Problem. Figures show a box-plot representation for different strategies and a bar representation for the percentage of winner solvers types.

In Figures D.2, D.3 and D.4, labels of the x-axis correspond to the following strategies:

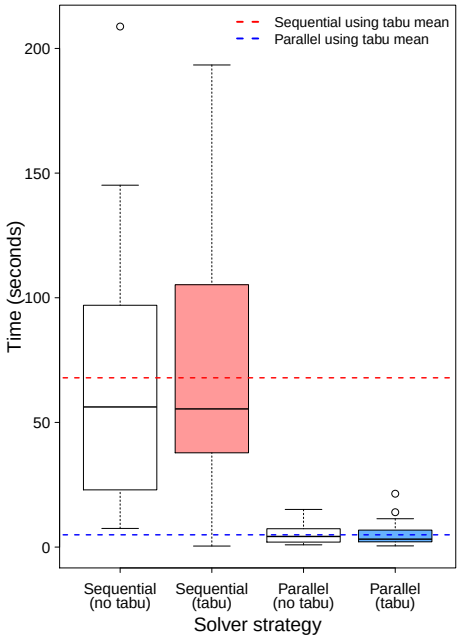
NC(noT): Non communication strategy without using tabu list
NC(T): Non communication strategy using tabu list
C1-1: Communicating solvers performing communication one to one
C1-n: Communicating solvers performing communication one to N

Figures D.5a, D.5b and D.5c, represent the percentage of winner solvers for each communication strategy, according to four different types:

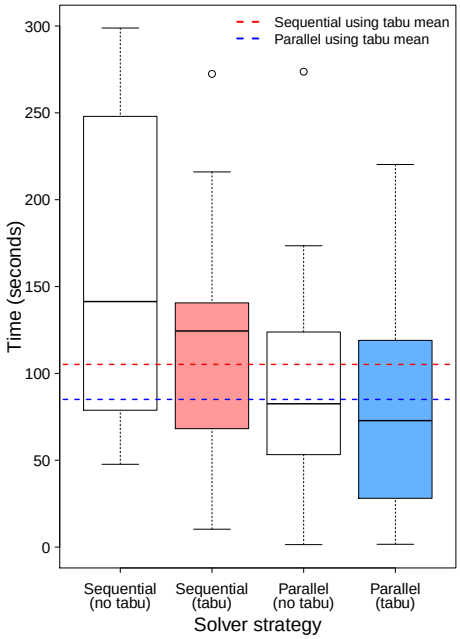
Receiver: Receiver solver wining thanks to the received information
Sender: Sender solver



(a) *GRP* 8-34



(b) *GRP* 10-55



(c) *GRP* 11-72

Figure D.1: Comparison between sequential and parallel runs to solve *GRP* using POSL

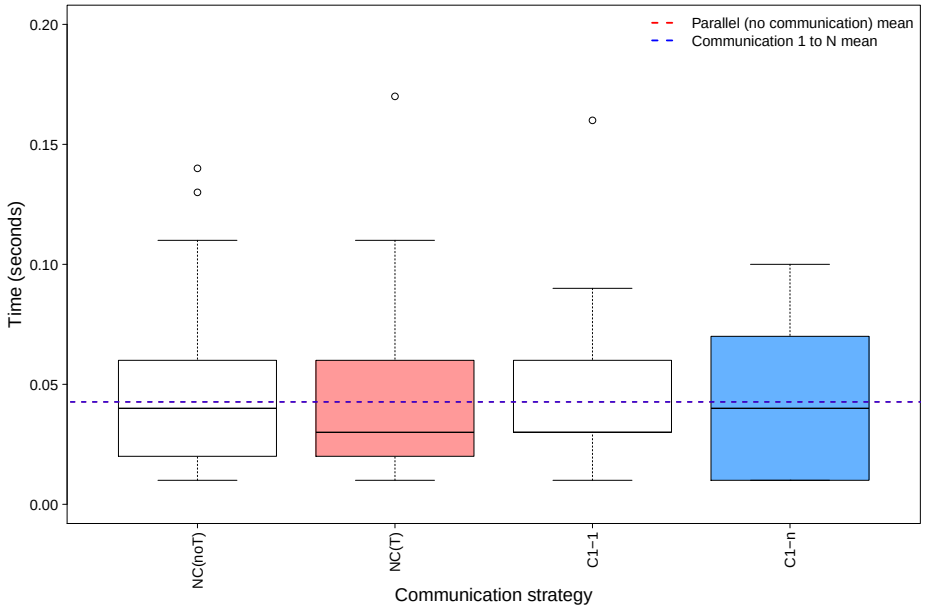


Figure D.2: Different communication strategies to solve *GRP* 8-34 using POSL

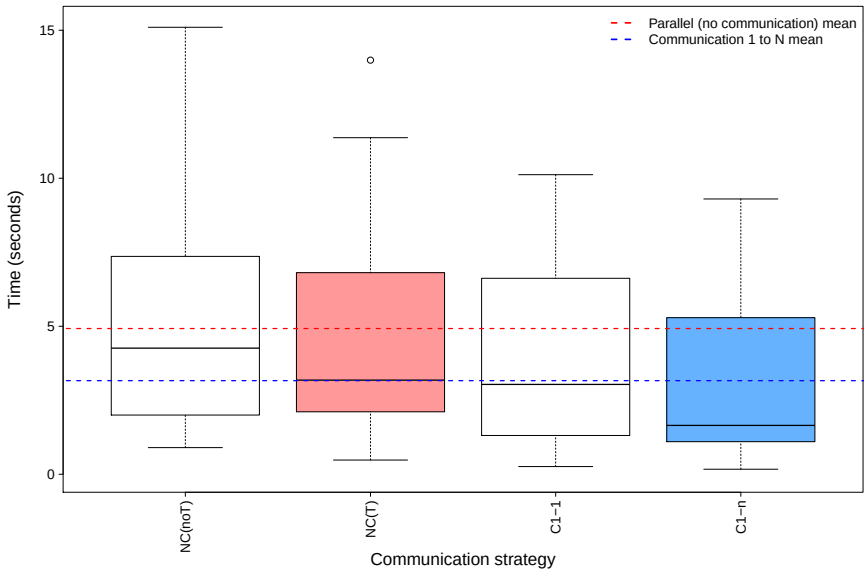


Figure D.3: Different communication strategies to solve *GRP* 10-55 using POSL

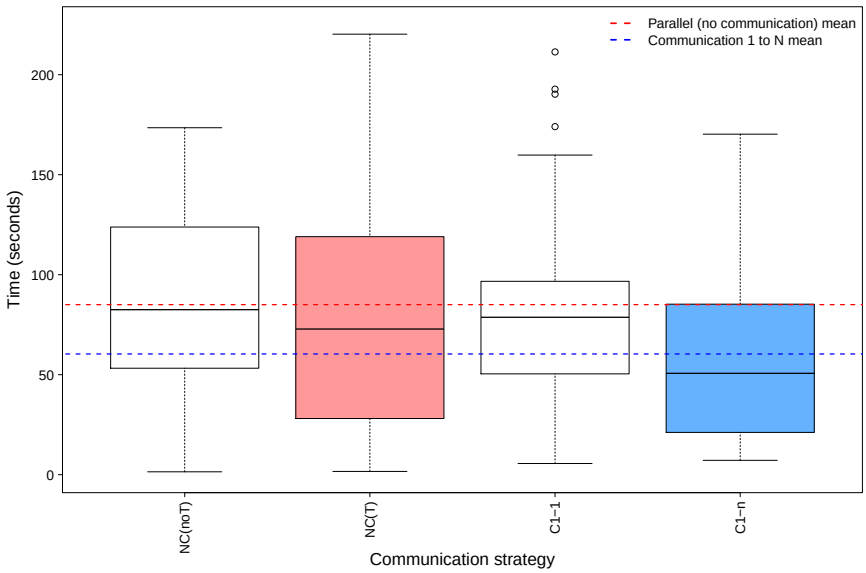
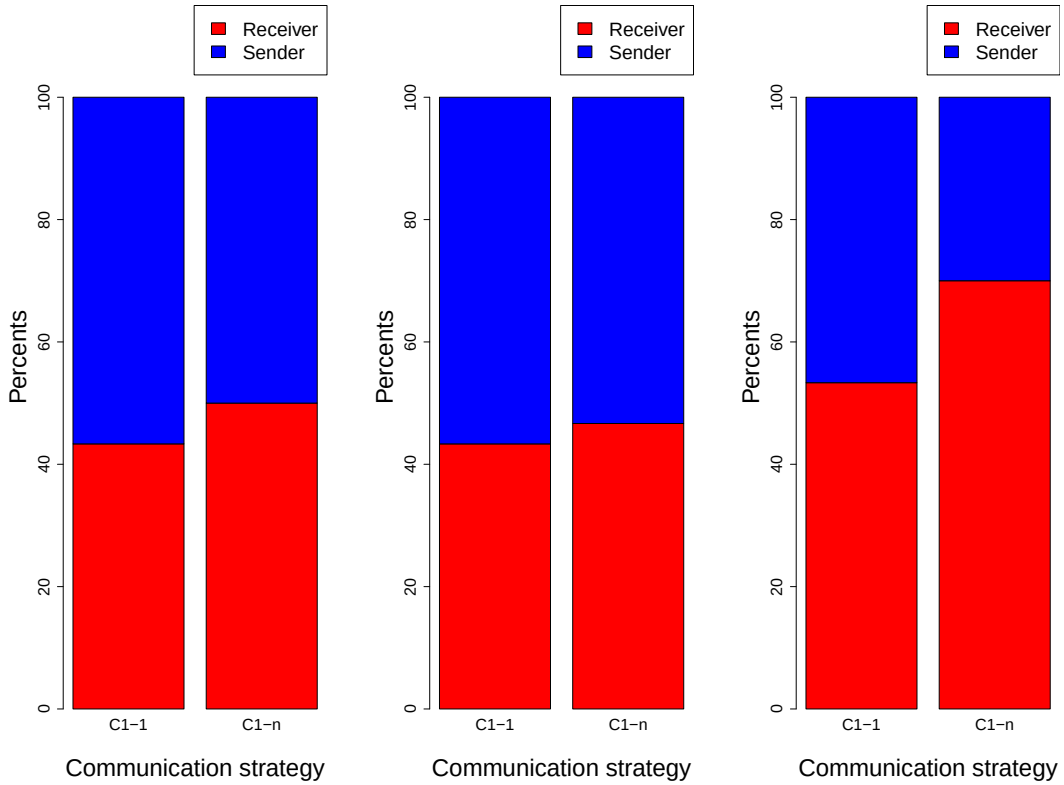


Figure D.4: Different communication strategies to solve *GRP* 11-72 using POSL



(a) *GRP* 8-34 **(b)** *GRP* 10-55 **(c)** *GRP* 11-72

Figure D.5: Solver proportion for each communication strategy to solve *GRP* using POSL