Solving Combinatorial Problems with Parallel Cooperative Solvers

Lei Duan, Samuel Gabrielsson, and J. Christopher Beck

Department of Mechanical and Industrial Engineering, University of Toronto
{lduan,samuel,jcb}@mie.utoronto.ca

Abstract. To exploit multi-core computing power, this paper presents parallel cooperative solvers for solving constraint satisfaction and optimization problems. Each solver owns the entire problem and exchanges partial solutions with other solvers. We applied Solution-Guided Multi-Point Constructive Search to the quasigroup-with-holes completion problem and Tabu Search to the quadratic assignment problem. Experimental results demonstrate that adding more solvers helps to improve the performance for both problems. We also introduce communication graphs and import policies to change solver cooperation. A combination of communication graph and import policy significantly impacts the performance, where the benefit from cooperation differs on the two problems: cooperation improves performance on the quasigroup-with-holes but not on the quadratic assignment problem.

1 Introduction

The multi-core architecture brings exciting opportunities for solving combinatorial problems. Today, a cluster formed out of several PCs can easily outstrip the performance of a decade-old supercomputer. Although hard combinatorial problems are notorious for being computationally prohibitive, larger problems have been successfully tackled, for example, by a series of fruitful exploitations of parallel computing [18, 12, 13, 16, 11]. Aligning with previous work on parallel search and distributed constraint programming, this paper presents parallel cooperative solvers, in which solutions, or partial solutions, are communicated to provide heuristic guidance. We experiment with Solution-Guided Multi-Point Constructive Search [2] for quasigroup-with-holes completion problems and with Tabu Search for quadratic assignment problems. Experimental results demonstrate that adding more solvers improves performance for both problems, although the performance gain relies on how solvers collaborate. The speedups and benefits from cooperation differ significantly on the two problems. The main contribution of this paper is an initial investigation of using parallel cooperative solvers to solve hard constraint satisfaction and optimization problems.

The crux of gaining performance with parallel computing lies in exploiting different aspects of parallelism in a problem. In parallel constructive search [12, 16], parallelism is exploited by solvers (also known as workers) searching different parts of the same search tree. While each worker explores a different subtree, all the workers are coordinated by a centralized control to be informed of a new search node and to avoid duplicating others' work. Essentially, the search space is partitioned and searched in parallel, leading to linear, and sometimes superlinear speedups [13]. In contrast, when the distributed nature of the problem (e.g. field sensor networks) or privacy concerns [4] make a centralized coordination impractical, the distributed constraint programming allows solvers (also known as agents) to divide the variable set and the constraints. When solving a problem, an individual agent attempts to satisfy internal constraints while communicating nogoods to resolve inter-agent constraints. Each agent operates asynchronously and searches in parallel [18, 11], leading to substantial increases in scalability [11, 14].

The cooperative solver approach we proposed is distinguished from the above as it relaxes the solver inter-dependence. Unlike parallel solvers sharing the same search tree, or distributed solvers sharing constraints, the cooperative solvers can be completely independent or fully collaborative. Communication is used to influence the search, as one solver's search can be guided by another's solution. Our hypothesis is that the best performance requires a good balance between guidance by an outside solution and searching on one's own. To this end, we introduce communication graphs and import policies to experiment with different points on the spectrum of collaboration versus independence. Although communicating solutions for guiding heuristic search has become commonplace for parallel combinatorial optimization [1], to our knowledge, no attempt has been made to guide constructive search for a constraint satisfaction problem using parallel cooperative solvers.

In Section 2, we provide a generic description of parallel cooperative solvers. The first part of the experiments (Section 3) applies Solution-Guided Multi-Point Constructive Search to the quasigroup-with-holes completion problem, and the second part (Section 4) presents results for the quadratic assignment problem using Tabu Search. Discussions and future work appear in Section 5, followed by conclusions in Section 6.

2 Communication and Solver Cooperation

A solver owns the entire problem, runs a search algorithm, and communicates solutions with other solvers. In a constraint satisfaction problem, a solution is a set of consistent variable assignments. We assume that a "better" solution has more assigned variables, and a complete solution has all the variables assigned and no constraints violated. In an optimization problem, a solution is always a complete and consistent assignment. A solver may also have an "elite" set holding a few high-quality solutions. An elite set enables a solver to keep track of high-quality solutions, either found by itself or obtained from other solvers.

In this paper, we propose a simple strategy for solver cooperation. We first describe how solvers exchange solutions according to a communication graph, then present two policies of treating solutions obtained from other solvers.

We specify that a solver can only communicate directly with its neighbours and use a communication graph to define a solver's neighbours. Each node represents a solver and each edge corresponds to bidirectional traffic of messages. The level of communication is indicated by the graph density, defined as the number of edges divided by the total number of edges of the complete graph. Take the four-solver communication graphs for example. Starting with density 0 (no edges) where no communication takes place and each solver attempts a problem independently, the next level up is a path connecting all the 4 nodes with density 0.5 (Figure 1(a)), then one more edge is added at a time, forming graphs of density 0.67 (Figure 1(b)), 0.83 (Figure 1(c)), and 1 (Figure 1(d)).[1] Our objectives are to find out: 1) whether adding more solvers helps to find a solution more quickly, and 2) how different communication graphs affect the performance.

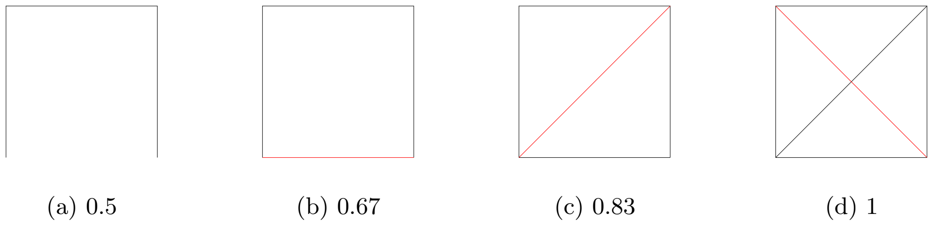

Fig. 1. Communication graphs for 4 solvers with different densities.

With communication, a solver periodically receives foreign solutions from its neighbours and decides how to import them into its elite set. An import policy selects one foreign solution at a time to replace the worst elite solution, if the selected solution is better than the worst elite. Otherwise, it will be discarded with all the other foreign solutions. A simple selection method is to always choose the best solution (referred to as import-best hereafter). In order to diversify search with different solutions, we introduce "import-softmax" which probabilistically chooses one foreign solution based on the idea of giving every solution a probability of being chosen but favouring solutions of better quality [3].

Given a set of N solutions $\{s_1, s_2, \ldots, s_N\}$ with their corresponding counts of unassigned variables for constraint satisfaction, or objective values for optimization (assuming the smaller the better) $\{a_1, a_2, \ldots, a_N\}$, the probability of choosing a solution s_i is mapped by the Boltzmann's distribution:

$$p(s_i) = \frac{\exp(Q_t(s_i)/\tau)}{\sum_{k=1}^{N} \exp(Q_t(s_k)/\tau)} \qquad (1)$$

[1] All the communication graphs for 4, 8 and 12 solvers used in the experiments are available at http://tidel.mie.utoronto.ca/parallelcoopsolvers.php.

where $Q_t(s_i) = a_i - \min_{k=1}^{N}\{a_k\}$, and $\tau = -(\max\{a_k\} - \min\{a_k\})$. (If $\tau = 0$, then $p(s_i) = 1/N$.) This mapping assigns a greater probability to a solution with fewer unassigned variables or a better objective value, ensuring that a better solution is more likely to be selected by import-softmax. It is easy to verify that $\sum_{i=1}^{N} p(s_i) = 1$.

An import policy has a local impact at each solver, while a communication graph impacts the global distribution of elite solutions. We call a combination of an import policy and a communication graph a cooperation configuration. How a configuration affects the solver performance will be studied in the experiments.

3 Experiment 1: SGMPCS for QWH

Solution-Guided Multi-Point Constructive Search (SGMPCS) has demonstrated strong performance on both constraint satisfaction problems [7] and optimization problems [2]. This notable success is mostly attributed to the combination of randomized restart and guiding search with elite solutions. The former can significantly boost the performance of constructive search [5] while the latter is the most effective mechanism in metaheuristics [17].

Pseudocode for the Solution-Guided Multi-Point Constructive Search is shown in Algorithm 1 (adapted from [7] for parallel cooperative solvers). The algorithm initializes a set, e, of elite solutions and then enters a while-loop. In each iteration, a set of foreign solutions received is examined. At most one foreign solution is inserted into the elite set, following the protocol described in the previous section. All other foreign solutions are discarded. Then a search is started from a randomly selected elite solution. If the best solution found, s, is better than the starting elite solution, s will replace the starting solution and will be sent to all the neighbours. Otherwise, the best elite solution will be sent.

Each individual search is limited by a certain number of fails. In multiple solvers, the algorithm will terminate when 1) a solver finds a complete solution, or 2) a solver proves that the problem has no solutions, or 3) each solver reaches its own maximum fail limit.

SGMPCS is identically configured according to the parameter settings in [7].

- *Initializing elite solution set* ($|e| = 8$) An elite solution is initialized by a chronological backtracking with smallest-domain for variable ordering followed by randomly choosing a value. Eight best solutions are retained out of 20 independent runs. Each run is restricted by a total of 1000 fails.
- *Bounding search* Each individual search is bounded by a polynomial fail limit: it is initialized to 32 and reset to 32 whenever a better solution is found, and it is increased by 32 when a search fails to improve a solution. The maximum fail limit is set to $2,000,000$ at each solver.
- *Guidance by elite solutions* Apart from initialization, every search is guided by an elite solution. A search tree is created using smallest-domain for variable ordering. Then a value is assigned as the same as it is in the elite solution, provided it is still in the domain of the variable. Otherwise, a value is randomly chosen and assigned.

Algorithm 1: SGMPCS for Parallel Cooperative Solvers

```
1 initialize elite solution set $e$
2 while not solved and termination criteria unmet do
      import a solution and replace worst($e$), if the imported solution $\notin e$ and is
      better than worst($e$)
3     $r$ := randomly chosen element of $e$
4     set fail bound, $b$
5     $s$ := search($r$, $b$)
6     if $s$ is better than $r$ then
7         replace $r$ with $s$
8         send $s$ out to neighbours
      else
9         send the best elite solution to neighbours
      end
  end
```

An $n \times n$ Quasigroup-With-Holes completion problem (QWH) [6] is a matrix where each row and each column is required to be a permutation of the integers $1, \ldots, n$. Since some of the matrix elements are left empty ("holes"), they are required to be filled with consistent values in order to find a complete quasigroup.

The performance evaluation includes the speedup of multiple solvers and the search cost, as measured by Concurrent Choice Points (CCPs), Aggregate Messages, and Aggregate CPU Time.

- The *concurrent choice points* take into account the dependency of search cost among solvers and are measured by the Cumulative Cost Algorithm [10]. The basic idea is that when a solver imports a solution, it also inherits the search cost of that solution. Then the solver will update its own choice point count, if the choice points of the imported solution are greater. At the end of the search, the concurrent choice points are taken as the biggest choice points among all the solvers.
- The *aggregate messages* measure the communication cost. Each message is composed of a solution and its choice point count. The aggregate messages sum up the total number of sent and received messages among all the solvers.
- The *aggregate CPU time* totals the CPU time of each solver upon termination. It comprises both time for search and time for communication.

The computing environment consists of 31 computing nodes. Each node has two 2.0 GHz AMD Dual Core processors with 4 Gb RAM running Red Hat Enterprise Linux 4. SGMPCS was implemented in ILOG Scheduler 6.0 with MPICH2 for communication.

Two sets of QWH instances are used. The first set comprises 10 order-33, 387-hole instances generated using the Gomes generator [6].[2] Each instance was attempted 10 times by 1, 4, 8, and 12 solvers, respectively. For a fixed solver

[2] It is available at: http://www.cs.cornell.edu/gomes/new-demos.htm

number, an instance was tried for all the configurations (i.e., combinations of all the communication graphs and the two import policies). This instance set is also used to study how configurations affect the performance. The second set is an existing benchmark comprising 18 instances with orders ranging from 30 to 70, which was also used in [6, 15, 7].

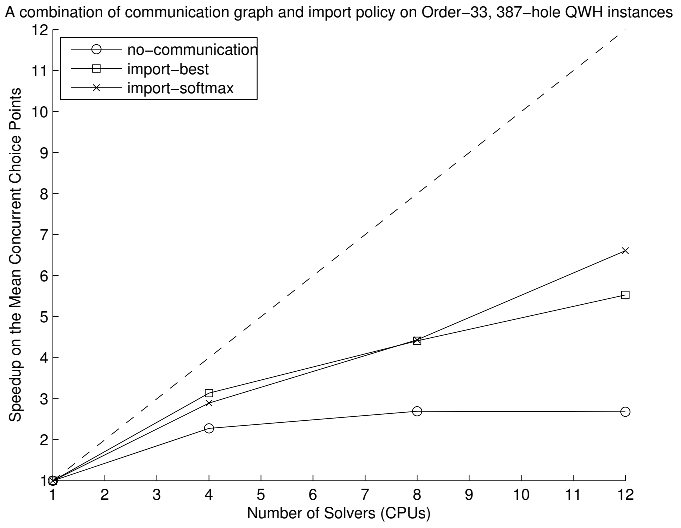

Fig. 2. Speedup on the mean concurrent choice points of 4, 8, and 12 solvers.

Figure 2 shows the speedup of using 4, 8, 12 solvers over a single solver running SGMPCS. The speedup is calculated as the mean choice points for a single solver divided by the mean concurrent choice points for multiple solvers with the best configuration. If a run fails to find a solution, the (concurrent) choice points are still used to calculate the mean. Figure 2 plots three curves: one without communication, one for communication based on the import-best policy, the other for communication with import-softmax. The figure clearly shows that adding more solvers helps to discover a solution more quickly, and cooperating solvers outperform their non-communicating counterparts. However, two important notes should be taken. First, the speedup is obtained by the best combination of communication graph and import policy. Therefore simply adding more solvers with an arbitrary configuration will not guarantee a speedup. In fact, there are quite a few cases in which communication hurts the performance, as shown below. Second, as can be seen from the dashed line in Figure 2, all the speedups are sublinear. For instance, the 12 solvers only achieved slightly more than a 6 times speedup. This is not surprising since solvers do not divide up the search space. Quite to the contrary, they may overlap a significant portion of the search space when intensifying search around a few elite solutions.

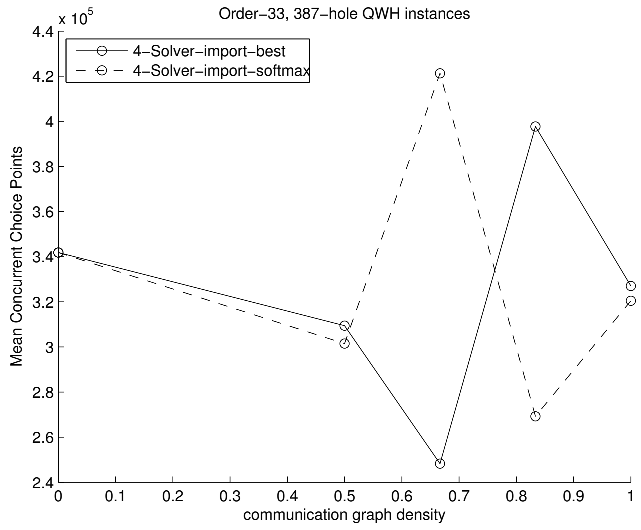

Fig. 3. Comparing the mean concurrent choice points across different communication graphs combined with import-best and import-softmax polices for 4 solvers.

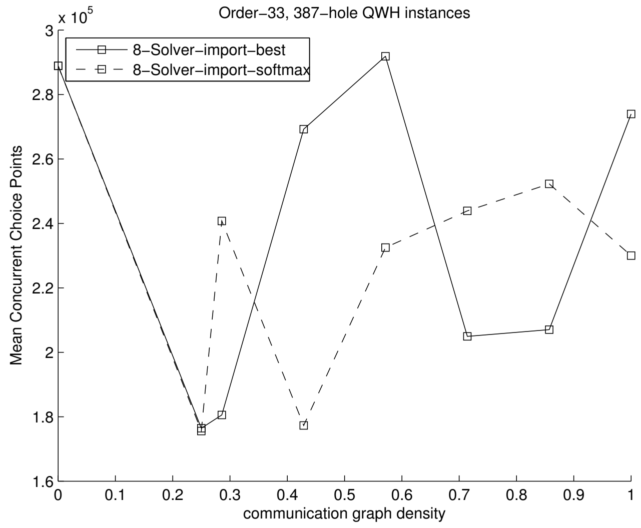

Fig. 4. Comparing the mean concurrent choice points across different communication graphs combined with import-best and import-softmax polices for 8 solvers.

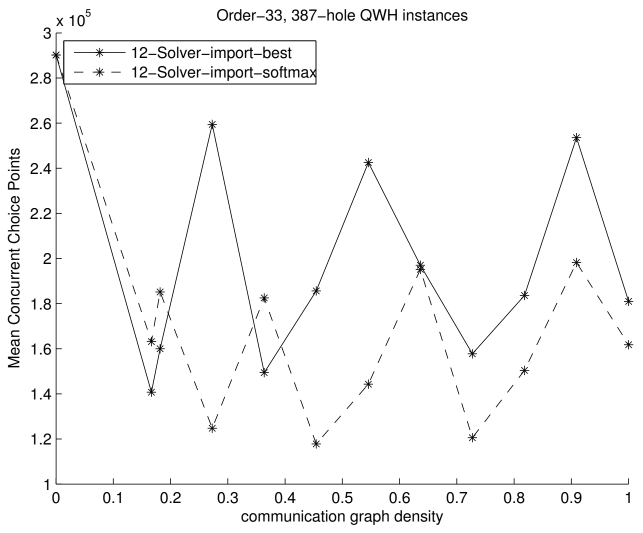

Fig. 5. Comparing the mean concurrent choice points across different communication graphs combined with import-best and import-softmax polices for 12 solvers.

Figures 3, 4, and 5 show the mean concurrent choice points over 10 runs across different communication graphs for 4, 8, and 12 solvers, respectively. Each figure demonstrates the impact of the two import policies. The communication graph, coupled with an import policy, significantly alters the performance. Contrary to the intuition that the curves would demonstrate a somewhat monotonic pattern, the concurrent choice points oscillate across communication graph densities as well as with import policies. Given a fixed number of solvers, the best result always occurs with communication, rather than without communication. For 4 solvers, the best CCP is obtained by the communication graph of 4 edges (density 0.67) with import-best. For 8 solvers, the best CCP is obtained by the graph of 7 edges (density 0.25) with import-softmax. For 12 solvers, the best CCP is obtained by the graph of 30 edges (density 0.45) with import-softmax. Interestingly, the best CCPs seem to be obtained by communication graphs with low or medium density.

Figure 6 shows the mean aggregate messages for 4, 8, and 12 solvers, respectively. The message volumes demonstrate roughly linear increases when more edges are added. Fewer messages result from lower graph densities as well as fewer concurrent choice points (i.e. finding a solution more quickly). Due to the space limit, Figure 7 shows the mean aggregate CPU time for 12 solvers only. Nevertheless, all the curves are in accordance with those in the CCP figures, showing that the CPU time is mainly consumed by searching for a solution instead of by communication.

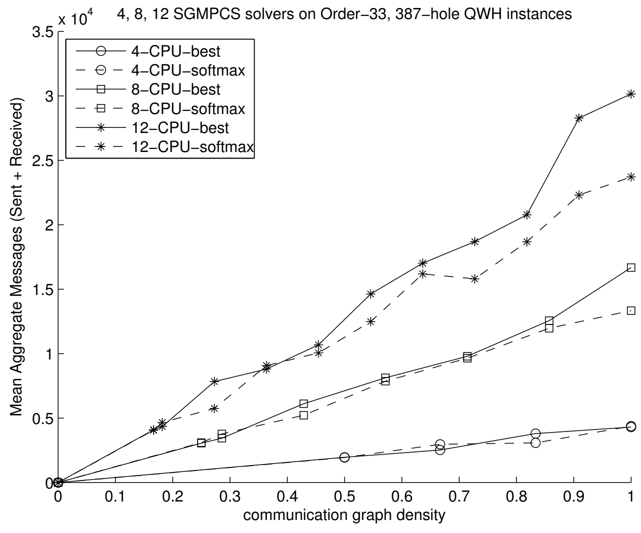

Fig. 6. Mean aggregate messages across different communication graphs combined with import-best and import-softmax polices for 4, 8, and 12 solvers.

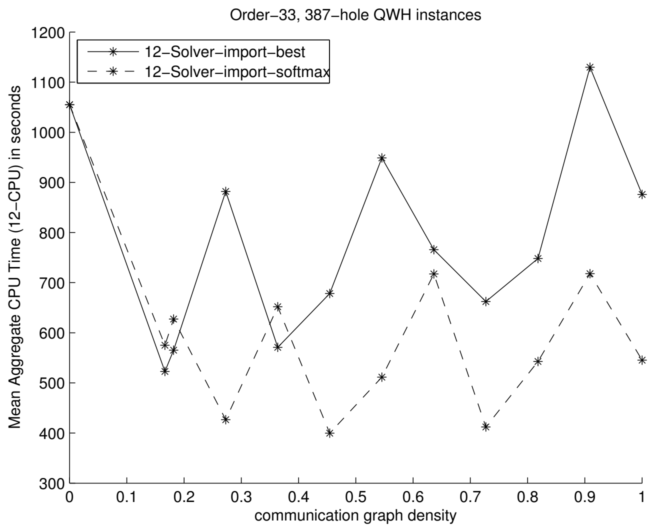

Fig. 7. Mean aggregate CPU time across different communication graphs combined with import-best and import-softmax polices for 12 solvers.

Table 1. Unsolved instances (out of 100) of order-33, hole-387 QWH problems

Number of Solvers	No Communication	Import-best	Import-softmax
1	19	-	-
4	5	3	3
8	1	1	1
12	1	**0**	**0**

Table 1 presents the number of unsolved instances based on the best configuration given a fixed number of solvers. For a single solver, the number of unsolved instances is 19 out of 100. Adding more cooperative solvers gradually reduces the number of unsolved instances. Finally, with 12 cooperating solvers, all the instances were solved in all the runs. Note that even though a run fails to find a solution, the (concurrent) choice points are still used to calculate the mean. Given the large number of unsolved instances for a single solver, the speedups shown in Figure 2 are an under-estimate.

Table 2. Comparing the mean (concurrent) choice points on QWH benchmark instances for 1, 4, 8, and 12 solvers with their best configurations. The bold entry indicates the best result on an instance. The unsolved is the number unsolved out of 10 runs.

		1 solver		4 solvers		8 solvers		12 solvers	
order	holes	unsolved	CPs	unsolved	CCPs	unsolved	CCPs	unsolved	CCPs
30	316	0	358	0	203	0	117	0	**62**
30	320	0	310	0	101	0	50	0	**35**
33	381	10	2025235	5	1905982	2	1474190	2	**888470**
35	405	0	192720	0	58637	0	50007	0	**32753**
40	528	6	1738612	0	**155179**	0	234865	0	165993
40	544	2	739356	0	152132	0	123276	0	**120126**
40	560	0	161053	0	64229	0	67287	0	**59260**
50	2000	0	1737	0	1716	0	1716	0	**1712**
50	825	10	2171697	10	2820974	**9**	**3299496**	9	3652465
60	1440	0	154308	0	96864	0	87643	0	**68414**
60	1620	0	199879	0	92284	0	78939	0	**74317**
60	1692	0	84941	0	59477	0	**43205**	0	43801
60	1728	0	76625	0	45740	0	44198	0	**43656**
60	1764	0	47068	0	**41745**	0	44491	0	44527
60	1800	0	50487	0	**42949**	0	45719	0	45707
70	2450	0	246042	0	117868	0	94622	0	**77291**
70	2940	0	36737	0	**27286**	0	49076	0	40918
70	3430	0	7581	0	3026	0	2990	0	**2967**

The last part of the QWH experiments compares the mean concurrent choice points for 1, 4, 8, and 12 solvers with their best configurations on a set of benchmark instances in Table 2. Each instance was attempted 10 times. Clearly, adding more solvers helps to solve more instances. For example, the two order-40 instances (528-hole and 544-hole) were solved in all the runs. With 8 and 12 solvers, the number of unsolved instances for the order-33-381-hole instance was brought to 2, down from 10 for a single solver. Also, parallel cooperative solvers significantly reduced the mean concurrent choice points, for example, more than 11 times fewer on the order-40-528-hole instance compared to a single solver.

4 Experiment 2: Tabu Search for QAP

In this section, we apply parallel cooperative solvers with tabu search to the Quadratic Assignment Problem (QAP). The main objective is to examine whether results from the constraint satisfaction problem case apply to an optimization problem as well.

A quadratic assignment problem [8] is mathematically formulated as follows. Given two matrices $A = (a_{ij})_{n\times n}$ and $B = (b_{kl})_{n\times n}$, and let the set Π be permutations of the integers from 1 to n. The goal is to find a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n)) \in \Pi$ such that

$$\min_{\pi\in\Pi} f(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} \qquad (2)$$

A is often interpreted as the flow matrix while B represents the distance matrix. An optimal solution minimizes the total flow cost between facilities. Large QAP instances are often tackled by heuristic methods such as Tabu Search. A review of recent advances for solving QAP appears in [9].

Tabu search tries to escape local optima by forbidding some recent moves recorded in a tabu list. In our implementation, an iteration of tabu search starts by evaluating all the 2-opt moves in the current neighbourhood. A 2-opt move selects two different elements and exchanges their positions in the permutation representing a solution. Tabu search selects a move in the following order: 1) if the best move in the tabu list leads to a better solution than the current best solution, the search will accept this move. 2) if the best tabu move does not improve the current best solution, then the search will choose the best non-tabu move. 3) if every move in the neighbourhood is in the tabu list, the search will choose the best tabu move, if it improves the present solution, or the move that has been kept in the tabu list for the longest time. Ties are broken randomly when choosing a move.

Each tabu search run by a solver is configured as follows.

- *tabu list size* A solver fixes its tabu list length by randomly choosing it from the set of $\{5, 7, 9, 11, 15, 17\}$.
- *one elite solution* A solver keeps track of the best solution found so far, either found by itself or imported from its neighbours.

– *Bounding search* Each individual tabu search is bounded by 500 iterations after no improvement. The maximum number of iterations is 10000.
– *Proportion of searching guided by elite solutions* After 500 iterations without improvement, a solver restarts search either from a randomly generated solution, or from the elite solution. The probability of restarting from the elite solution is set to 0.5.
– *communication* A solver sends the best solution to its neighbours every 50 iterations and imports a solution at the end of each individual search. The imported solution will replace the current elite solution, if it is better.

Tabu search is implemented in C++ with MPICH2 for communication. The same computing environment as for the QWH experiments was used. The QAP instances are a set of recently released problems based on microarray layout.[3] Each instance was attempted 5 times with the same communication graphs as SGMPCS for QWH, combined with the import-softmax policy.

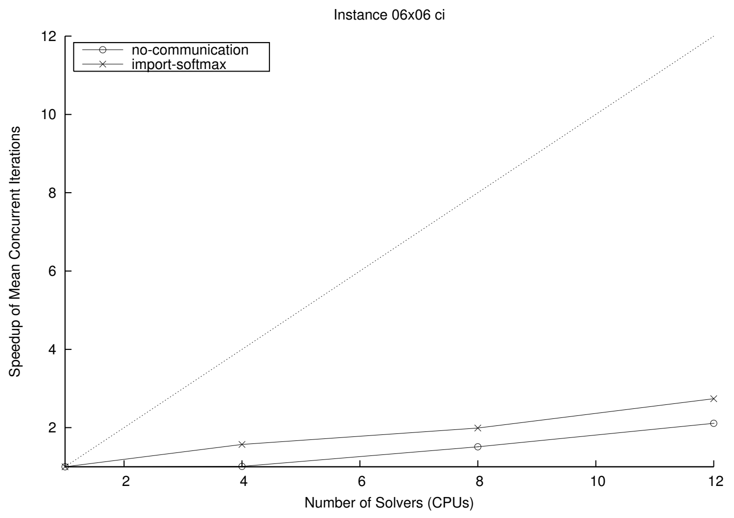

Fig. 8. Speedup on the mean concurrent iterations of 4, 8, and 12 solvers.

The speedup on the mean concurrent iterations is calculated as follows. 1) The best solution ever found is identified and a very close range within the best objective value, e.g. 0.1%, is targeted. 2) For each run, the iteration at which the objective value first reaches the target range is recorded. If a run fails to reach the range, the maximum iteration (10000) is used instead. 3) the iterations from 2) for a single solver are averaged over all the runs, then divided by the corresponding mean iterations for multiple solvers. Figure 8 shows the speedup for the 0.1% target range within the best overall solution on the chip size 6×6

[3] Details at http://gi.cebitec.uni-bielefeld.de/comet/chiplayout/qap/index.html

conflict index instance. Other instances show similar speedup results. Similar to the QWH case (Figure 2), adding more solvers helps to reach a high-quality solution more quickly, and communication with the best configuration outperforms non-communication. However, the speedup develops more slowly: the 12 solvers achieved a mere 2.7 speedup with the best cooperation configuration and non-communication only 2.1.

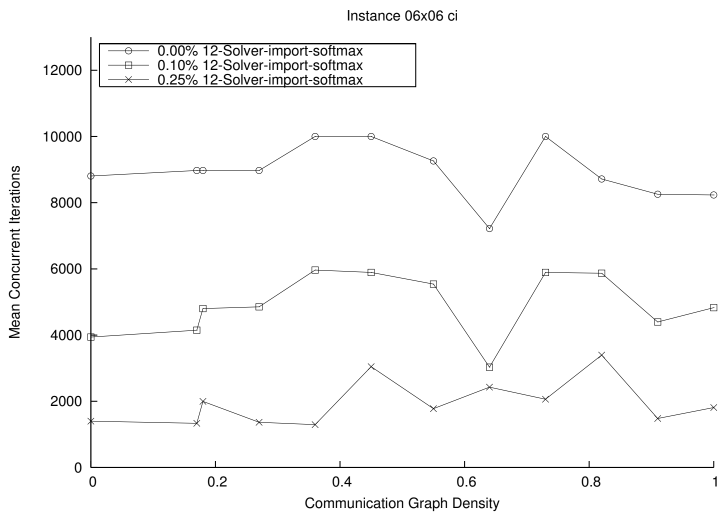

Fig. 9. Comparing the mean concurrent iterations across different communication graphs for 12 solvers. The best solution found has an objective value of 168611971. The target ranges are 0%, 0.1%, and 0.25% away from this objective value.

Figure 9 presents the mean concurrent iterations for 0%, 0.1%, and 0.25% target ranges across different communication graphs for 12 solvers. The difference from the QWH results becomes more pronounced. While cooperation clearly boosts solver performance on QWH, the high-quality QAP solutions, marked by the small target ranges, predominantly fall into non-communicating solvers, except in two cases at the 42-edge communication graph (density 0.64) for the 0% and 0.10% target ranges. Nevertheless, the cooperative solvers outperform several best known solutions on this QAP instance set, as shown in Table 3.

5 Discussion and Future Work

Similar to constructive search, the work-stealing in parallel constructive search [12, 16] carefully coordinates different workers to guarantee no repeated search. But this restriction hampers search from freely following heuristics. In contrast, cooperative solvers, resembling local search, are free to follow heuristics but may overlap on search space explorations. This paper raises an analogous question:

Table 3. Comparing objective values on the microarray-layout border length minimization (bl) and conflict index minimization (ci) QAP instances using 12 cooperative solvers with the best configuration to the best known results. The relative error is calculated as $\frac{Obj - Obj_B}{Obj_B} \times 100\%$ where Obj is the objective value obtained by the solvers and Obj_B is the best known. A bold entry indicates a new best solution. The wall time is the duration between the start of executing the solvers and the end at which all the solvers terminate.

Instance	Best Known	Cooperative Solvers	Relative Error (%)	wall time (sec.)
6×6 bl	$3,296$	$3,296$	0.00	299.9
7×7 bl	$4,564$	**4,560**	**−0.09**	933.5
8×8 bl	$6,048$	**6,040**	**−0.13**	1782.3
9×9 bl	$7,644$	$7,648$	+0.05	4307.2
10×10 bl	$9,432$	$9,452$	+0.21	7693.7
11×11 bl	$11,640$	$11,676$	+0.31	10907.6
12×12 bl	$13,832$	$13,848$	+0.12	25743.6
6×6 ci	$169,016,907$	**168,611,971**	**−0.24**	305.1
7×7 ci	$237,077,377$	**236,613,631**	**−0.20**	1054.1
8×8 ci	$326,696,412$	**326,376,790**	**−0.10**	1787.2
9×9 ci	$428,682,120$	$430,224,089$	+0.36	3217.6
10×10 ci	$525,401,670$	$528,471,212$	+0.58	7118.2
11×11 ci	$658,317,466$	$662,898,977$	+0.70	14854.3
12×12 ci	$803,379,686$	$809,244,786$	+0.73	24683.1

Work-stealing is systematic but restrictive, while cooperative solvers are free to explore search space but may duplicate search. For very large problems, is it rewarding to give up the systematicity of work-stealing in exchange for better performance using cooperative solvers in parallel search? We aim to answer this question by a direct comparison between work-stealing and cooperative search on large combinatorial problems in the future.

Another area for future work is to enable a solver to dynamically change its neighbours and to adjust its import policy, thus overcoming the hurdle of having to experiment with a large number of combinations. For example, several solvers can start with a fully connected communication graph and end up with an empty one, should independent solving turn out to be more desirable for a particular problem. Also, solvers can adapt themselves when more solvers join or some solvers leave. This would require a solver to autonomously evaluate the relevance of an imported solution and even a neighbour using a more sophisticated criterion, and to become more proactive in procuring heuristic guidance.

6 Conclusion

In this paper, we presented parallel cooperative solvers in which each solver communicates solutions. Experimental results demonstrated that adding more solvers helps to improve the performance for quasigroup-with-holes completion problems with SGMPCS and for quadratic assignment problems with Tabu

Search. We also introduced communication graphs and import policies to change the way in which solvers cooperate. Experimental results showed that solvers benefit from the heuristic guidance through collaboration on the quasigroup-with-holes completion problem, but not as much on the quadratic assignment problem. We observed that a cooperation configuration has a significant impact on search performance. Therefore further investigations are needed to understand how communication affects the performance.

References

1. E. Alba, editor. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005.
2. J. C. Beck. Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *Journal of Artificial Intelligence Research*, 29:49–77, 2007.
3. J. L. Bresina. Heuristic-biased stochastic sampling. In *AAAI*, pages 271–278, 1996.
4. B. Faltings. *Handbook of Constraint Programming*, chapter 20, Distributed Constraint Programming, pages 699–729. Elsevier, 2006.
5. C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *AAAI*, 1998.
6. C. P. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of Computational Symposium on Graph Coloring and Generalizations*, 2002.
7. I. Heckman and J. C. Beck. An empirical study of multi-point constructive search for constraint satisfaction. In *Proceedings of the Third International Workshop on Local Search Techniques in Constraint Satisfaction*, 2006.
8. T. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.
9. E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176:657–690, 2007.
10. A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proceedings of 3rd Workshop on Distributed Constraint Reasoning (AAMAS 2002)*, 2002.
11. P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.
12. L. Perron. Search procedures and parallelism in constraint programming. In *CP*, 1999.
13. L. Perron. Practical parallelism in constraint programming. In *CPAIOR*, 2002.
14. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.
15. P. Refalo. Impact-based search strategies for constraint programming. In *CP*, 2004.
16. C. Schulte. *Programming Constraint Services : High-Level Programming of Standard and New Constraint Services*. Springer, 2002.
17. J.-P. Watson, A. E. Howe, and L. D. Whitley. Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem. *Computers and Operations Research*, 33(9):2623–2644, 2006.
18. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.