New Benchmarks for Static Task Scheduling on Homogeneous Multiprocessor Systems with Communication Delays

Tatjana Davidović
Mathematical Institute,
Serbian Academy of Science and Arts,
Kneza Mihaila 35, 11000 Belgrade, Yugoslavia
tanjad@crt.umontreal.ca

Teodor Gabriel Crainic
Departement management et technologie,
Université du Québec à Montréal and
Centre de recherche sur les transports, Université de Montréal
theo@crt.umontreal.ca

January 8, 2003

Abstract

Scheduling tasks on multiprocessor systems is a very active research area with numerous papers addressing the many different variants of the problem. Yet, there appears to be a lack of adequate benchmark problem instances. In this paper, we propose a large set of benchmark graphs for the Multiprocessor Scheduling Problem with Communication Delays. The proposed benchmark problems have known optimal solutions and cover a broad spectrum of characteristics: multiprocessor architecture, number of processors, number of tasks, density of inter-task communications links. We use these benchmark problem instances to analyze the performance of several constructive heuristics and may thus emphasize the dependency of constructive heuristic solutions on different problem parameters related to both task graph characteristics and multiprocessor architecture.

Key words: Multiprocessor systems, task scheduling, communication delays, benchmark problem instances

Résumé

L'allocation et l'ordonnancement de tâches sur les processeurs d'un ordinateur multiprocesseurs constitue un domaine très actif de recherche et de développement et de multiples travaux ont été dédiés aux diverses variantes du problème. On observe, toutefois, un manque sérieux de problémes tests permettant de analyses comparatives poussées. Nous proposons un ensemble de 700 problèmes tests avec solutions optimales connues pour le problème d'allocation de taâches avec delais de communication. Les problèmes proposés différent selon le type d'architecture du système, le nombre de processeurs, le nombre de tâches, la densité des liens de communication inter-tâches. Nous avons utilisé cet ensemble de problèmes pour analyser le comportement de plusieurs méthodes heuristiques de construction de solution, ce qui nous permet de souligner la sensibilité de ces méthodes aux variations dans le caractéristiques des problèmes reliées à la fois au graph à allouer et au systéme multi-processeur.

Mots-clefs: Ordinateurs multi-processeurs, Allocation et ordonnancement de tâches, delais de communication, problèmes tests

1 Introduction

The problem of the efficient use of multiprocessor systems, addressed by scheduling program modules (tasks) to processors, has been studied for over forty years now [9]. The problem is known to be NP-hard in its most general form [8, 24]. There are only a few special cases that can be solved optimally in polynomial time [3, 10, 25]. For all the other problem variants, numerous heuristics have been proposed to efficiently obtain suboptimal solutions [7, 12, 16, 21, 22]. See [2] for a recent survey.

Performance of heuristics are often difficult to analyze and compare, however, even when they address the same scheduling problem variant. Not only theoretical performance measures are difficult to come by, but even the experimental analysis is hampered by the fact that most papers propose new test instances and report efficiency results for these examples only. The utilization of acknowledged sets of benchmark problem instances (e.g., the OR Library: http://mscmga.ms.ic.ac.uk/info.html) to report computational results contributes to alleviate this issue. Such sets of benchmark problem instances are not available for all classes of scheduling problems, however.

We are particularly interested in the problem of scheduling dependent tasks to homogeneous multiprocessor systems with processors connected in an arbitrary way. Moreover, we take into account the time spent for transferring data between tasks allocated to different processors. Despite the interest of the problem and the work already accomplished in the area, very few benchmark problem instances have been proposed. Ribeiro, Porto, and Kitajima [**?**, 19, 20] proposed a set of problems of relatively small sizes, but report scheduling results only for completely connected heterogeneous multiprocessor systems. Tobita and Kasahara [23] recently presented a large set of problem instances, but communications and multiprocessor architecture are not explicitly considered. In fact, according to the authors' knowledge, Kwok and Ahmad [14] proposed the only existing benchmarks closely related to the problem addressed in this paper. It appears, however, that these problems are not very challenging, shown in Section 3. There is thus the need for a more complete and challenging set of benchmark problem instances.

The objective of this paper is to propose new sets of benchmarks with known optimal solutions and different characteristics in terms of the type of multiprocessor architecture, number of processors, number of tasks to schedule, and the density of the precedence/communication task graph. We also analyze the scheduling results obtained by several constructive heuristics for these problems. This allows us not only to confirm that there are "easy" and "hard" graphs for each heuristic (which is an already well known fact), but also, and most importantly, to explain the dependency of scheduling results on task graph and multiprocessor characteristics. The proposed benchmark instances should thus also prove useful in testing meta-heuristics that are increasingly proposed to address scheduling problems (e.g., [4, 5, 13]), and that often proceed by searching some solution space and applying a constructive heuristic to obtain the corresponding schedule

and objective function value.

The paper is organized as follows. The considered scheduling problem is described in Section 2. Section 3 describes the problem set introduced by Kwok and Ahmad [14] and the new benchmarks that we propose. The results of the performance analysis of constructive heuristics applied to the new benchmarks are reported in Section 4, while Section 5 concludes the paper. The complete set of figures displaying the scheduling results is included in the Appendix.

2 Problem Formulation

The tasks to be scheduled are represented by a *Directed Acyclic Graph (DAG)* [4, 11, 22] defined by a tuple $\mathcal{G} = (T,\ E,\ C,\ L)$, where $T = \{t_1, \ldots, t_n\}$ denotes the set of tasks; $E = \{e_{ij} \mid t_i,\ t_j \in T\}$ represents the set of precedence/communication edges; $C = \{c_{ij} \mid e_{ij} \in E\}$ denotes the set of edge communication costs; and $L = \{l_1, \ldots, l_n\}$ represents the set of task computation times (execution times, lengths). The communication cost $c_{ij} \in C$ denotes the amount of data transferred between tasks t_i and t_j if they are executed on different processors. If both tasks are assigned to the same processor, the communication cost equals zero. The set E defines precedence relations between tasks. A task cannot be executed unless all of its predecessors have completed their execution and all relevant data is available. Task preemption and redundant executions are not allowed in the problem version considered in this paper.

The multiprocessor system $\mathcal{M}$ is assumed to contain p identical processors with their own local memories. Processors communicate by exchanging messages through bidirectional links of the equal capacity. The architecture is modeled by a *distance matrix* [4, 7]. The element $(i,\ j)$ of the distance matrix $D = [d_{ij}]_{p \times p}$ is equal to the minimum distance between the nodes i and j. In the following, the minimum distance is calculated as the number of links along the shortest path between two nodes. It is obvious that the distance matrix is symmetric with zero diagonal elements.

The scheduling of DAG $\mathcal{G}$ onto $\mathcal{M}$ consists of determining the index of the associated processor and starting time instant for each of the tasks from the task graph in such a way as to minimize some objective function. The usual objective function (that we use in this paper as well) computes the completion time of the scheduled task graph (also referred to as *makespan, response time* or *schedule length*). The starting time of a task t_i depends on the completion times of its predecessors and the amount of time needed to transfer the associated data from the processors executing these predecessors to the processor that has to execute the task t_i. Depending on the multiprocessor architecture, the time that is spent for communication between tasks t_i and t_j can be calculated as

$$comm_{ij}(l,\ k) = c_{ij} * d_{lk} * ccr,$$

where it is assumed that task t_i will be executed on processor p_l, task t_j on processor p_k, and ccr represents the Communication-to-Computation-Ratio defined as the ratio between the transfer time of a unit amount of data and the time spent for performing a single computational operation. Note that this definition is characterizes the multiprocessor architecture and is thus different from that of the CCR parameter of Kwok and Ahmad [13, 14] (defined as the ratio between total communication and total computation times) that corresponds to a characterization of the task graph. If $l = k$ then $d_{lk} = 0$ implying that $comm_{ij}(l, k) = 0$.

3 Benchmarks Description

Kwok and Ahmad [14] performed an impressive work by collecting 15 scheduling algorithms proposed in the literature and generating several sets of test instances to compare them. Since different algorithms usually introduce different assumptions, it is almost impossible to compare them all. Consequently, the authors divided the scheduling algorithms into five groups and compared the algorithms within groups. Similarly, test instances were partitioned into groups and not all scheduling algorithms were applied to all groups. The authors also proposed criteria for performance evaluation and comparison of scheduling algorithms.

The most interesting set of test instances is the one containing task graphs with known optimal solution. Kwok and Ahmad generated two groups of such test examples. The first set contains 36 graphs: for each of the 12 DAG sizes corresponding to a number of tasks n ranging from 10 to 32 by increment of 2, three (3) graphs were generated with three different values for CCR parameter. Optimal solutions for these small examples were determined by using an A^* enumeration algorithm [1]. The heuristic results reported displayed 1% to 8% deviations from the optimum. The second group of test examples consists of 30 instances with the preselected optimal solution for completely connected multiprocessor systems. For these examples, n ranges from 50 to 500 by the increment 50 and, again, for each n, three different values were used for the CCR parameter. Scheduling results were reported for 11 scheduling algorithms and 2% to 15% average deviation was registered. Known optimal solutions were obtained only in few cases.

We run the CPES algorithm on the second group of test examples. This algorithm is a modification of the DLS scheduling heuristic [22], in the sense that static priorities based on the Critical Path (CP) method are assigned to each task. The critical path does not involve the communication times as in the case of the method reported in [12]. Rather, the calculation of the length of the critical path is based only on task execution times l_i. The "ES" stands for Earliest Start, meaning that for each processor p_k, the starting time $st(t_i, p_k)$ of task t_i is calculated and the task is allocated to the processor with the smallest associated starting time value [4].

The results for scheduling the benchmark instances with known optimal solution by using the CPES algorithm were surprising: Optimal schedules were obtained for all 30 instances. This would mean that CPES is "the best heuristic in the world" and our experience [5, 6] is that the results obtained by CPES can deviate by more than 100% from the optimum. Analyzing the task graphs of these test instances, we noticed that they are all very sparse graphs, with edge (connection) density ρ varying from 8% to 15%. It than becomes clear that these graphs contain almost independent tasks. Moreover, differences in multiprocessor architecture is not accounted for. Thus, for each n, there is only one multiprocessor system (with fixed number of completely connected processors) associated to the 3 test instances of that size.

In this paper we propose a new, large set of randomly generates test instances with known optimal solution that overcomes some of these limitations. The 700 problem instances we propose display different characteristics in terms of multiprocessor system architecture, number of tasks, and connection density. We selected 7 different multiprocessor system configurations defined by the connection architecture and the number of processors p: Four hypercubes of dimensions 1, 2, 3, and 4 (i.e., 2, 4, 8, and 16 processors), and three mesh configurations of 6, 9, and 12 processors. For each of the 7 system configurations, 100 problem instances are generated by combining the number of tasks n, 10 values from 50 to 500 by increment of 50, and the connection density ρ, 10 values from 0 (independent tasks) to 90% of the maximum allowed density ρ_{max} of the corresponding task graph.

The graphs are generated similarly to Kwok and Ahmad [13, 14]. A number m (set here to 10) graphs is generated for each combination of multiprocessor architecture, defined by the number of processors p and the distance matrix $D_{p\times p}$, number of tasks n, and length of the optimal schedule SL_{opt}. All m graphs contain the same number of tasks, with the same task durations but with different edge densities ρ. First, the optimal schedule is generated in such a way that each processor j executes approximately the same number of tasks x_j (with 10% allowed deviations). The task durations are determined randomly, using a uniform distribution (mean equal to SL_{opt}/x_j and 10% deviation). All processors are completing the execution at the same time (SL_{opt}) and there are no idle time intervals between tasks. Tasks are then numbered according to their starting times: the first task on the first processor obtains number 1, the first task on the second processor obtains number 2, and so on. p+1 will be the number of the second task with the smallest starting time (regardless of the index of the corresponding processor); p+2 will be associated to the task that is the next one to begin its execution (the task could be on the same processor if the previous one has a small duration); etc.

Finally, precedence relations between tasks are defined by adding edges to the graph and assigning communication loads to these edges. The maximum allowed density ρ_{max} is calculated first, because there cannot be an edge from task t_i to t_j if $st(t_i, p_k) + l_i > st(t_j, p_l)$). Then, for each $i = 0, ..., m-1$, the number of edges corresponding to the

density $i/m * \rho_{max}$ is calculated and new edges are added randomly to the task graph to reach that number. The communication amount along each edge (edge weight) is calculated by the following rule:

$$c_{ij} = \begin{cases} \infty, & \text{if } k = l, \\ (st(t_j, p_l) - (st(t_i, p_k) + l_i))/d_{kl}, & \text{otherwise.} \end{cases}$$

i.e., when tasks are executed on the same processor, the amount of data exchanged can be arbitrary large, otherwise, the communication amount is defined by the interval between the completion time of the first task and the starting time of the second one divided by the distance between the corresponding processors. The task graph obtained in this way is written in the output file, and becomes the starting point for building the instance with the next higher density (only a few new edges are needed to be added).

The lengths of optimal schedules for each number of tasks n in the DAG are given in Table 1. These lengths do not depend on the values for p and ρ. The proposed set of problem instances can thus be used for scheduling on completely connected architectures, as well as when communication delays are not relevant.

n	50	100	150	200	250	300	350	400	450	500
SL_{opt}	600	800	1000	1200	1400	1600	1800	2000	2200	2400

Table 1: Optimal schedule lengths for random task graphs

The zip file `Bench_opt.zip`, which can be downloaded from the following web address `http://www.mi.sanu.ac.yu/~tanjad/Bench_opt.zip`, contains the task graph examples and the `readme.txt` file. File names are `ogra<n>_<r>_<p>.td`, where $< n >$ should be substituted with the number of tasks in that file, $< r >$ with the edge density, and $< p >$ with the number of processors for which the optimal solution is given.

In each file, data are written in the following format:

- The first row contains the number of tasks n;
- The next n rows contain the data for task $i = 1, \ldots, n$: index i, duration t_i, number of successors n_{succ}, the list of n_{succ} pairs s_j c_{ij} representing the index of the j^{th} successor and corresponding communication amount.

Other than the test instances with known optimal solutions, the zip-file `Bench.zip` with 180 completely random task graphs can also be found at the same web address. The task graph files are written in the same format, with file names `t<n>_<r>_<i>.td`, where n and r have the same meaning as in the previous case, while i is the index of graph with same n and r value (there are 6 graphs for each (n, r) pair). The graphs in that

zip-file do not depend on multiprocessor architecture, they can be scheduled to arbitrary multiprocessor systems but, since optimal solutions are not known, the solution quality cannot be predicted.

4 Scheduling results

In this section we report and analyze results of scheduling the random task graphs with known optimal solution we introduce by using CPES, as well as several other constructive scheduling heuristics. The quality of the heuristic solutions is examined relative to parameters related to both task graph characteristics and multiprocessor architectures: number of tasks n, edge density ρ, number of processors p, interconnection architecture, and communication delay.

Other than CPES, we implemented the PPS [15], LBMC [21], and DC [4] heuristics. LBMC is a two-step heuristic method based on balancing the load (LB) of the processors and minimizing communications (MC): The task allocation to processors is performed in the first step, while the actual schedule (i.e., definition of the order among tasks and of the starting time for each task) is performed during the second step. PPS is also based on the same two-step specification: First, tasks are assigned to processors according to the Preferred Path Selection (PPS) rule, and schedules are determined in the second step. For CPES and DC, the steps are defined according to the list-scheduling ideas: Tasks are first sorted according to some priority scheme. Then, each task is assigned to the processor selected by the ES rule for CPES and by an ES improvement of the sequential schedule (DeClustering) for DC. Since task priorities, as well as assignment rules, can be defined in different ways, several variations of these heuristic methods are possible (for example, combination of Largest Processing Time (LPT) priorities with LB, maximum communication with ES, etc.). It is not our intention, however, to compare and classify scheduling heuristics but to show that the proposed set of benchmarks is representative. We thus focus on these four heuristics.

A limited number of the scheduling results for graphs from the set of 700 random test instances are illustrated in this section. Complete results are summarized in the Appendix (actual figures can be obtained from the authors). For all figures, unless otherwise stated, the results correspond to CPES applications.

We first analyzed the influence of the edge (communication) density on the solution quality. Since the results for each p are similar, we display in Figures 1 and 2 the results for $p=8$ to illustrate the observations connected to this analysis. A curve shows for each n, the change in the solution deviation when ρ grows. Therefore, each graph displays 10 curves. Two cases are considered for the multiprocessor architecture with a fixed number of processors p: Connected according to a given interconnection structure (mesh

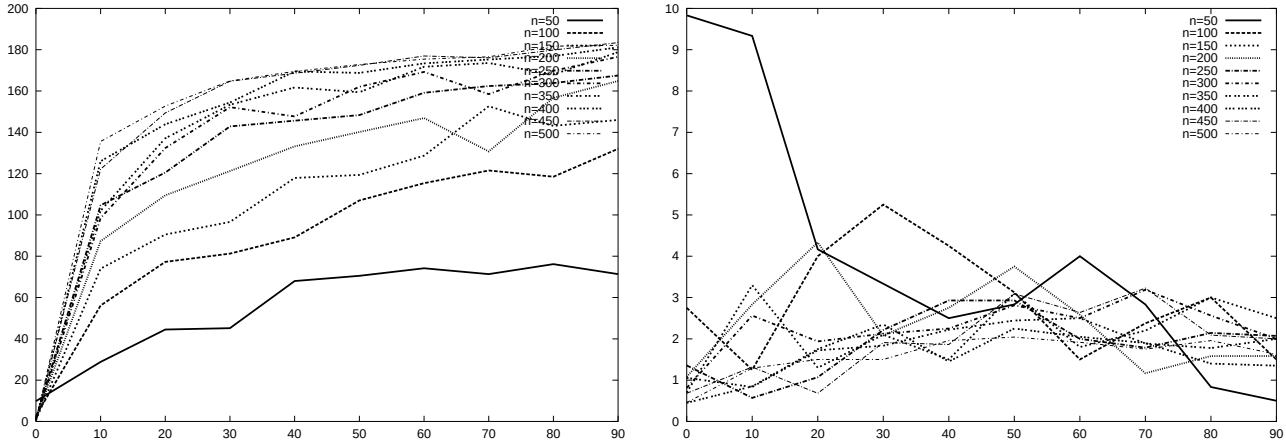

Figure 1: Solution deviations for 8-processor hypercube, with and without communication delays

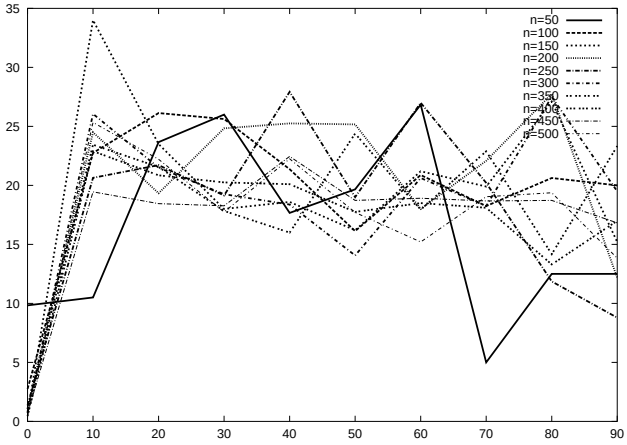

Figure 2: Solution deviations for completely connected 8 processors with communication delays

or hypercube) or completely connected, i.e. there is a direct communication link between each two processors. We also distinguish between cases when communication delay is significant and when it can be neglected. In the latter case, the interconnection network does not play any role. Consequently, three cases exist (and three graphs are displayed) for each p: Particular interconnection network with and without communication delays and complete interconnection network with communication delays. An exception is the case $p = 2$ since the particular and the complete connections are the same; Therefore, only two cases have to be analyzed.

As Figures 1 and 2 illustrate, with a small number of exceptions, the deviation from optimality grows with ρ growing when communication time is significant. In addition, the processor connection is very important for the scheduling process: Deviations when processors are not completely connected can be large, an order of magnitude larger

than for completely connected processors. Moreover, for completely connected processor systems, the deviation does not depend on ρ that much, but the growing tendency can still be observed, especially for higher number of processors. When one can neglect the communication time, the deviations are very small.

These results show that sparse task graphs are easy to schedule with CPES. This explains why all benchmarks from [14] were scheduled optimally and emphasizes the need for a more complete set of benchmark problem instances.

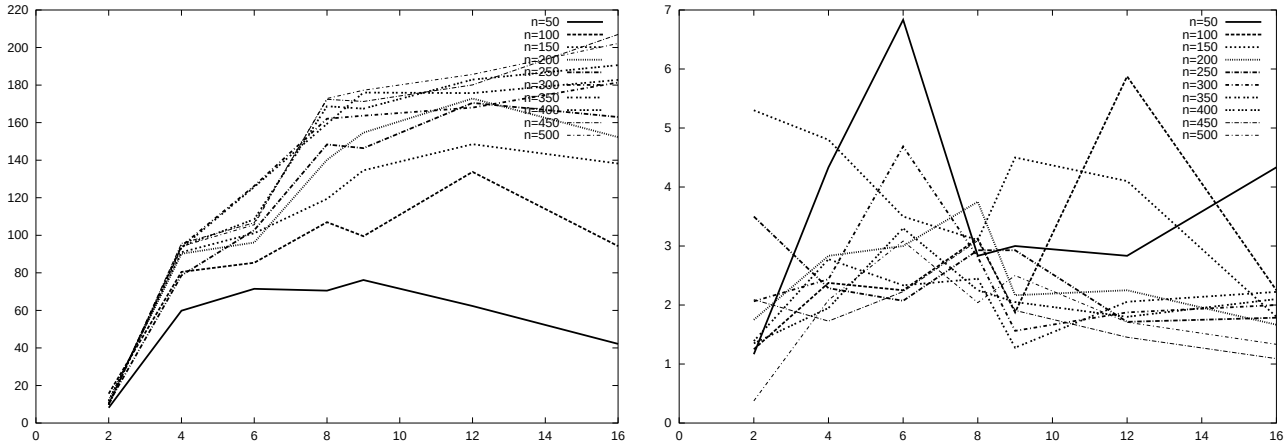

Figure 3: Solution deviations for task graphs with $\rho = 50$, with and without communication delays, scheduled onto incomplete networks

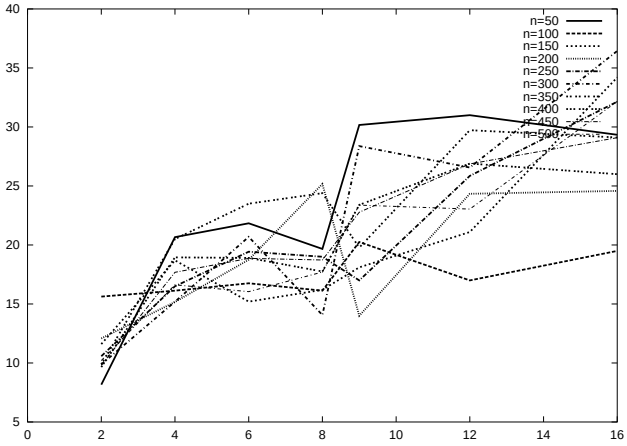

Figure 4: Solution deviations for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

Next, we examined the influence of the number of processors on the scheduling results. It is natural to expect to be easier to schedule for systems with smaller number of processors than for larger multiprocessor systems. Data dependencies can indeed prevent the efficient exploitation of a large number of available processors. Sometimes, the

number of processors can be even two large, resulting in very significant time spent on data transfers.

Figures 3 and 4 illustrate the deviations in the schedule lengths obtained by CPES, according to the number of processors for graphs with $\rho = 50\%$. The results support our assumption: when communication times are significant, the deviation grows with the number of processors. If communication times are irrelevant, tasks can be packed in a better way. For completely connected processors, the deviations are still growing, but less rapidly. Here is the second reason for the optimality of the CPES scheduling results on the problem instances from [14]: For each n, only one value for p is given (although p is growing with n).

The complete scheduling results obtained by using CPES, for all values of p and ρ, are given in the Appendix. From this analysis, summarized in the previous paragraphs, it appears that it is easier to schedule sparse task graphs than more dense ones. Moreover, better result should be expected when the number of processors is relatively small. We believe such hard or easy graphs exist for all scheduling heuristics and that the set of problem instances we propose is sufficiently large to include challenging problems in most cases.

To examine this hypothesis, we run the LBMC, PPS, and DC constructive heuristics on the same set of task graphs. The corresponding results are also given in the Appendix, starting with Figure 49. Here, we only include figures relative to the same cases used for the CPES to illustrate and support our conclusion. See Figures 5 to 16. The results show that PPS and DC encounter the same difficulties as CPES in scheduling dense task graphs. Moreover, the DC method does not distinguish between different types of processor connections. LBMC seems to have even more difficulty in addressing sparse task graphs, except for independents tasks where it acts like the LPT method. Deviations also tend to grow with the number of processors, except for LBMC and PPS for which the 4-dimensional hypercube seems to be easier to handle than the 12-processor mesh in the case when communications are significant.

The complete set of test instances is very large and it may not be necessary to use all the examples. According to our analyses, however, at least one of the two parameters p and ρ should be varied. This can reduce the size of benchmark set by an order of magnitude (either 70 or 100 examples need to be evaluated, depending on which parameter is varied, p or ρ). Based on the experiments with the four constructive heuristics, we concluded that p and ρ are equivalent, in the sense on the insight their variation provides to the user. It is up to the user to decide which one to vary. We believe this conclusion holds for other methods as well.

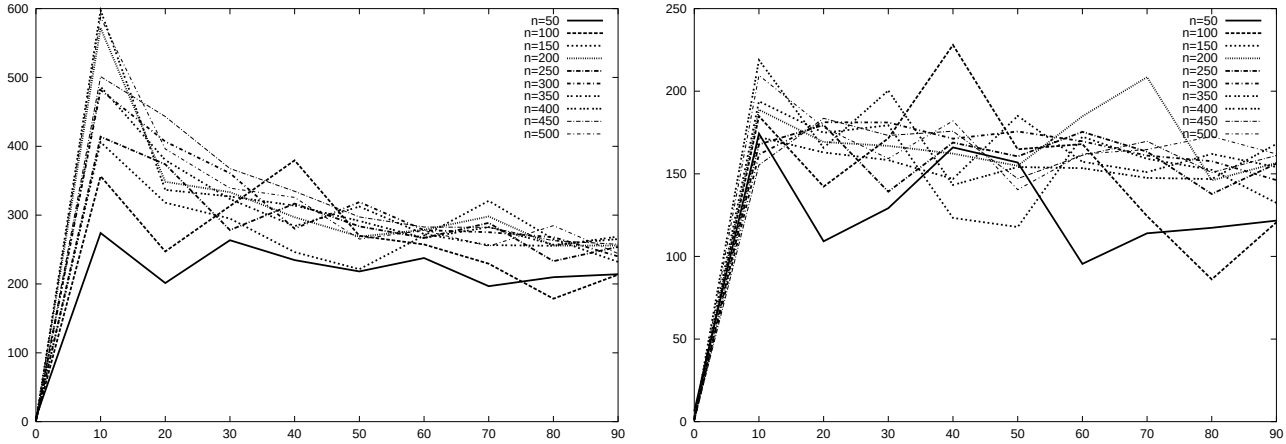

Figure 5: LBMC Solution deviation for 8-processor hypercube, with and without communication delays

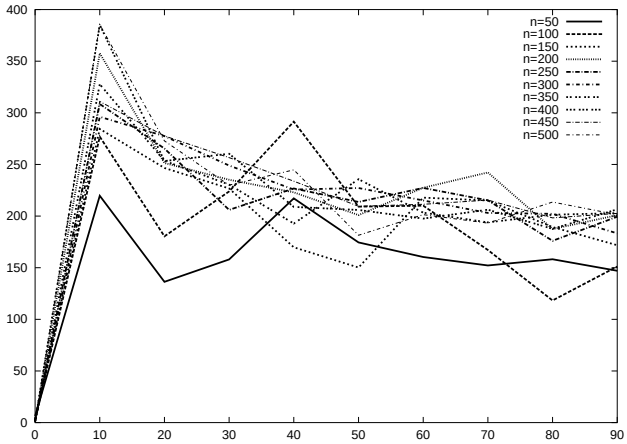

Figure 6: LBMC Solution deviation for completely connected 8 processors with communication delays

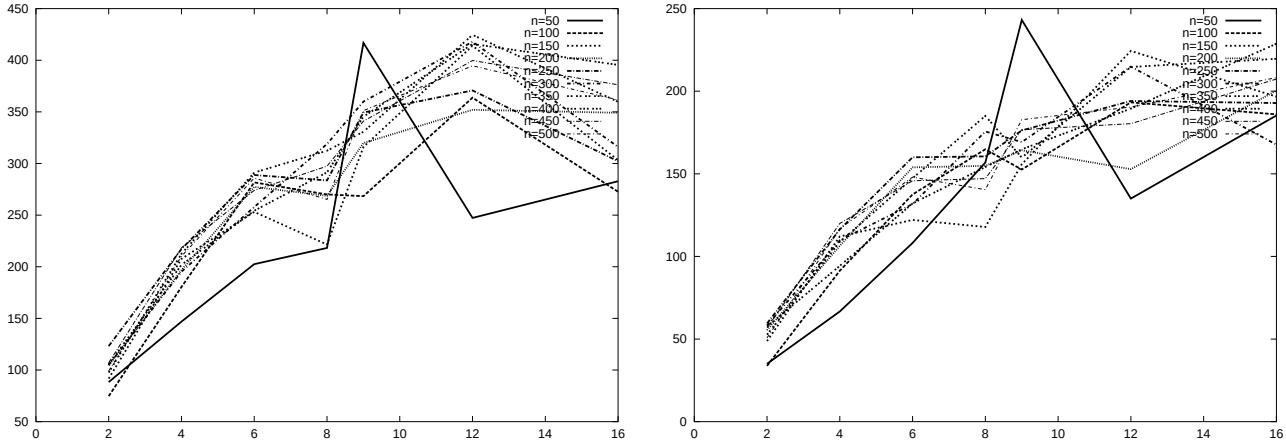

Figure 7: LBMC Solution deviation for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

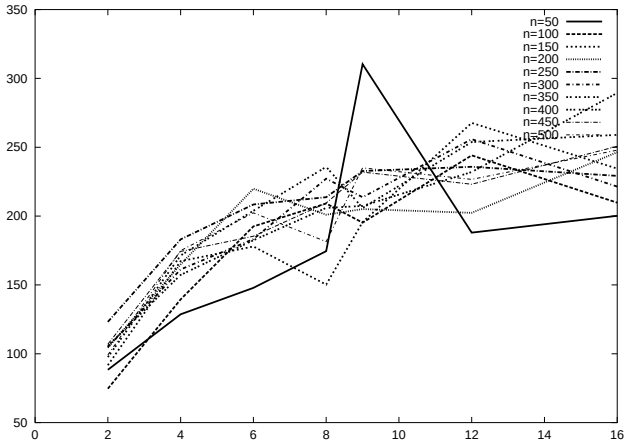

Figure 8: LBMC Solution deviation for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

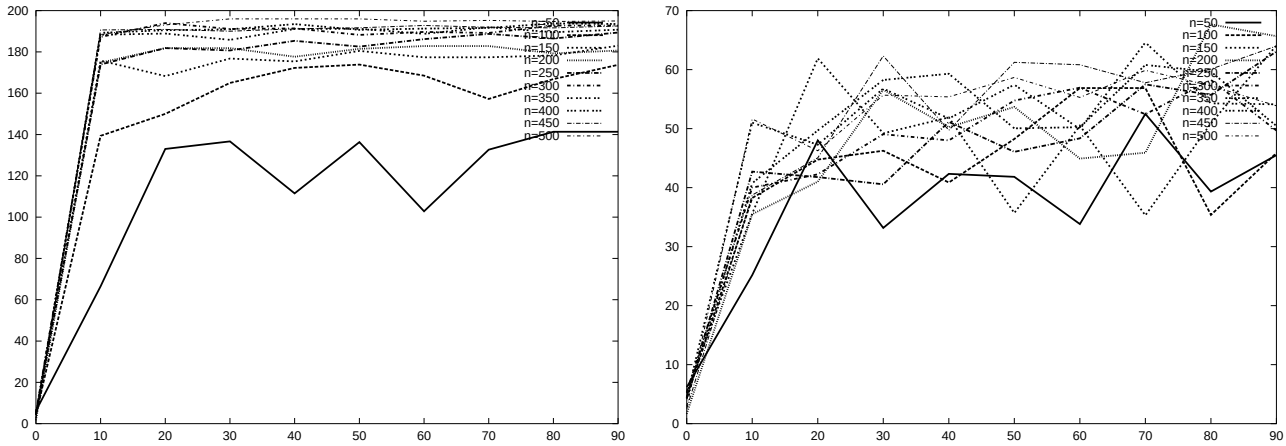

Figure 9: PPS solution deviation for 8-processor hypercube, with and without communication delays

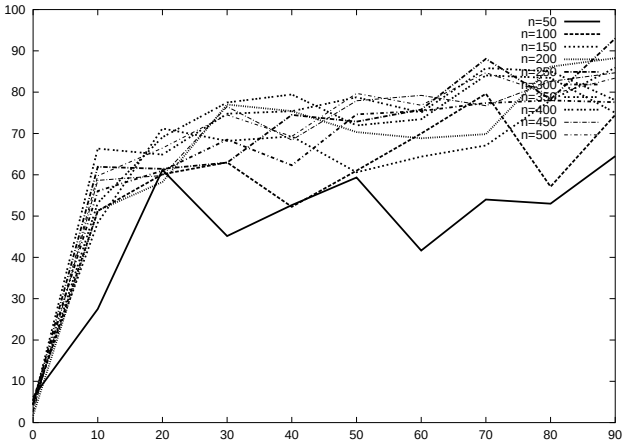

Figure 10: PPS solution deviation for completely connected 8 processors with communication delays

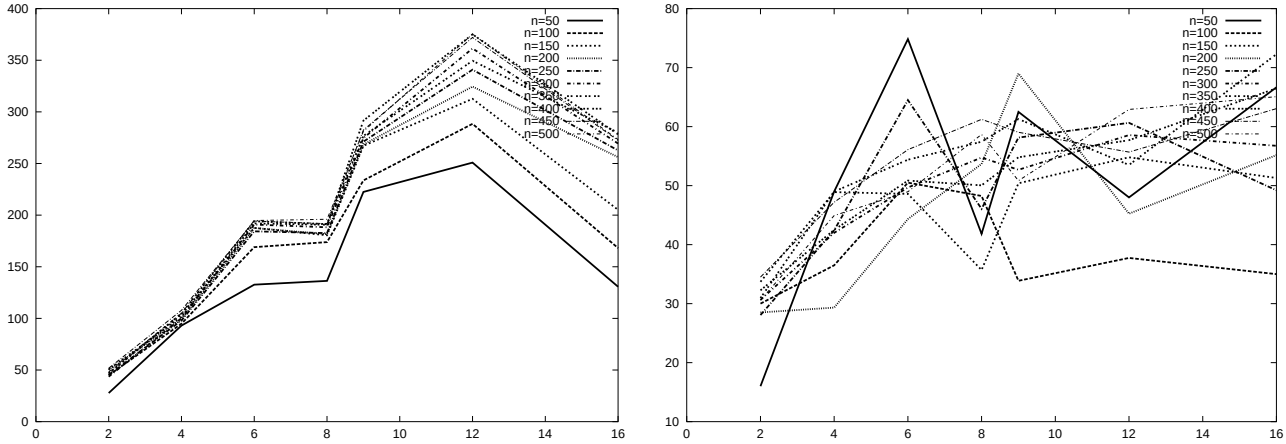

Figure 11: PPS Solution deviation for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

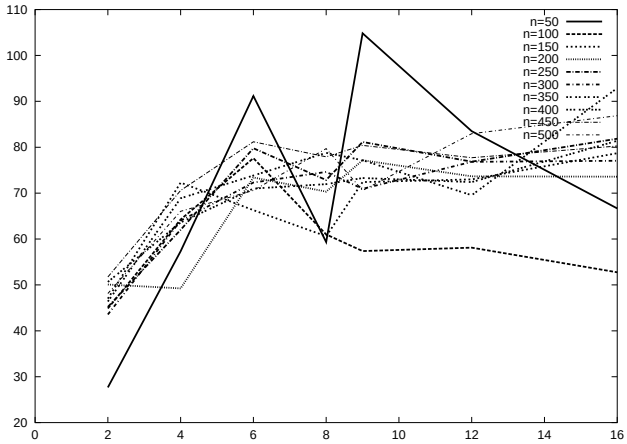

Figure 12: PPS Solution deviation for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

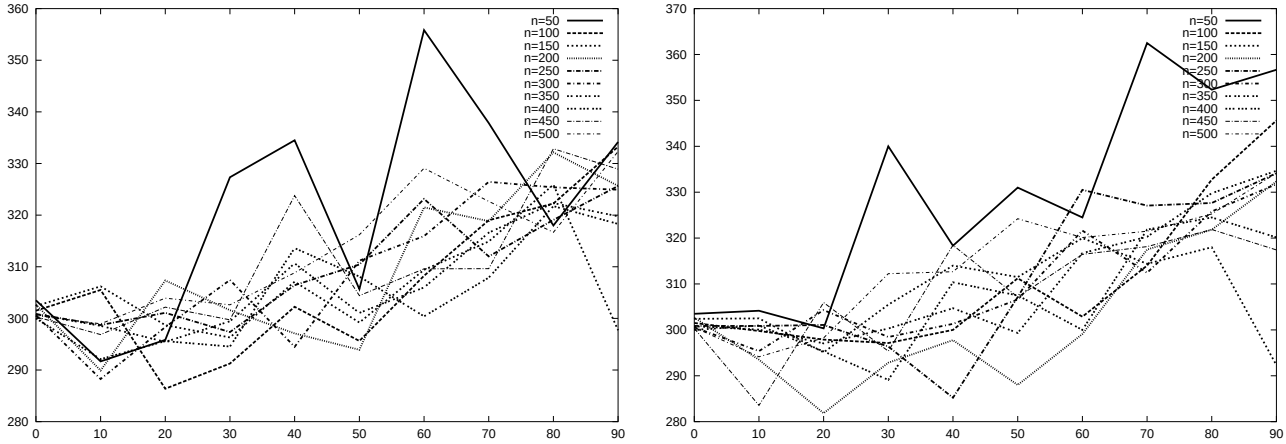

Figure 13: DC solution deviation for 8-processor hypercube, with and without communication delays

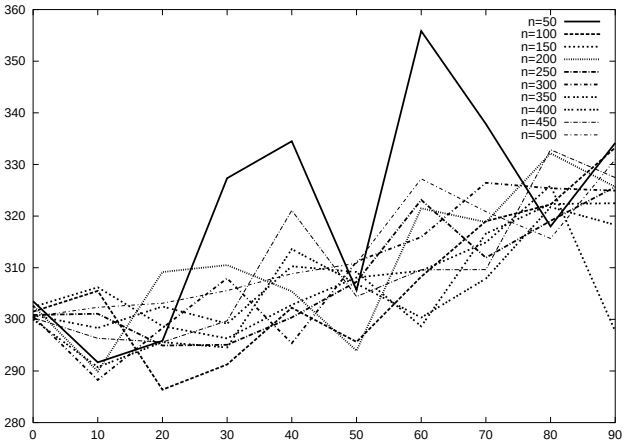

Figure 14: DC solution deviation for completely connected 8 processors with communication delays

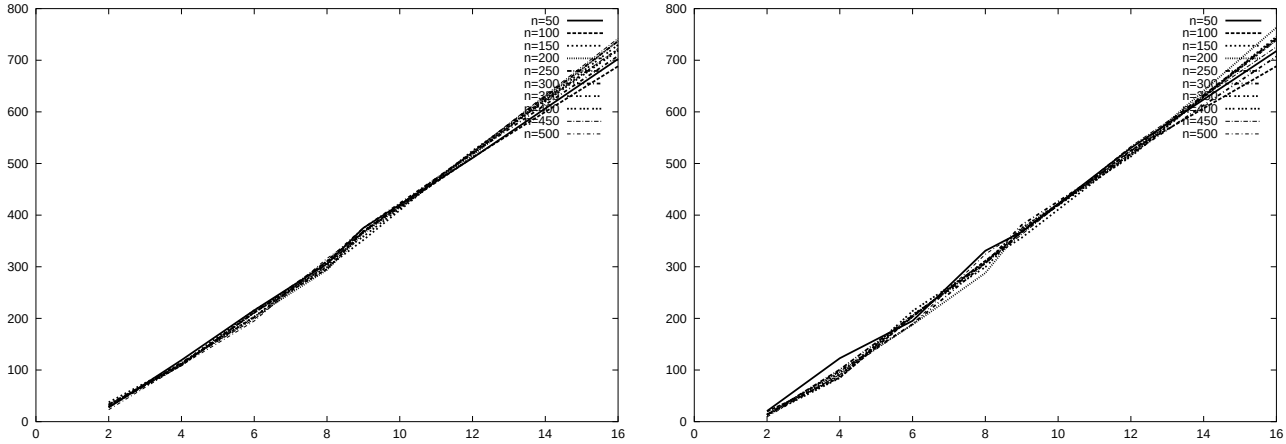

Figure 15: DC Solution deviation for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

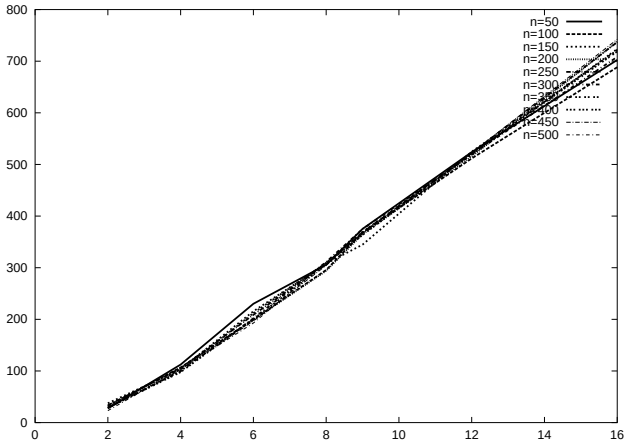

Figure 16: DC Solution deviation for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

5 Conclusion

In this paper we propose a new set of benchmark problem instances with known optimal solutions for scheduling dependent tasks onto different multiprocessor architectures, not necessarily completely connected, taking into account communication delays.

We analyzed the dependency of the deviation of heuristic solution from the optimal one on several parameters: task graph density, communication delays, number of processors and connections between them. These parameters play a significant role in the quality of heuristic solutions. We concluded that communication delays are significantly degrading the performance of the scheduling heuristics. Regarding the multiprocessor architecture, it appears easier to schedule onto completely connected processors. For the other two parameters, number of processors and task graph density, at least one of them should be varied in order to objectively evaluate the efficiency of a particular scheduling heuristic.

Most of the problem instances we propose are difficult, in particular for constructive scheduling heuristics and, as such, should offer a greater challenge than the problems currently available to scheduling heuristics and meta-heuristics, in their sequential or parallel forms.

Acknowledgments

This project was partially supported by NSF of Serbia, grant no. 1583, under project "Mathematical optimization models and methods with applications". Funding for this project has also been provided by the Natural Sciences and Engineering Council of Canada, through its Research Grant programs, and by the Fonds F.C.A.R. of the Province of Québec.

References

[1] Ahmad, I. and Kwok, Y.-K. Optimal and Near-Optimal Allocation of Precedence-Constrained Tasks to Parallel Processors: Defying the High Complexity Using Effective Search Technique. In *Proceedings 1998 International Conference on Parallel Processing*, 424–431, 1998.

[2] Blazewicz, J., Drozdowski, M., and Ecker, K. Management of Resources in Parallel Systems. In J. Blazewicz, K. Ecker, B. Plateau, D. Trystram, editors, *Handbook on Parallel and Distributed Processing*, 263–341. Springer-Verlag, New York, 2000.

[3] Coffman Jr., E.G. and Graham, R.L. Optimal Scheduling for Two-Processor Systems. *Acta Informatica*, 1:200–213, 1972.

[4] Davidović, T. Exhaustive List-Scheduling Heuristic for Dense Task Graphs. *YUJOR*, 10(1):123–136, 2000.

[5] Davidović, T., Hansen, P., and Mladenović, N. Scheduling by VNS: Experimental Analysis. In *Zbornik Jug. Simp. o Operacionim Istraživanjima, SYM-OP-IS 2001*, 319–322, Beograd, 2001.

[6] Davidović, T., Hansen, P., and Mladenović, N. Variable Neighborhood Search for Multiprocessor Scheduling Problems with Communication Delays. In *Proceedings MIC'2001, 4^{th} Metaheuristics International Conference*, 737–741, Porto, Portugal, 2001.

[7] Djordjević, G. and Tošić, M. A Compile-Time Scheduling Heuristic for Multiprocessor Architectures. *The Computer Journal*, 39(8):663–674, 1996.

[8] Garey, M.R. and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completness*. W. H. Freeman and Company, 1979.

[9] Hu, T.C. Parallel Sequencing and Assembly Line Problems. *Operations Research*, 9(6):841–848, 1961.

[10] Krishnamoorthy, V, and Efe, K. Task Scheduling with and without Communication Delays: A Unified Approach. *European Journal of Operational Research*, 89:366–379, 1996.

[11] Kwok, Y.-K. and Ahmad, I. Bubble Scheduling: A Quasi Dynamic Algorithm for Static Allocation of Tasks to Parallel Architectures. In *Proceedings 7^{th} IEEE Symposium of Parallel and Distributed Processing (SPDP'95)*, 36–43, Dallas, Texas, USA, 1995.

[12] Kwok, Y.-K. and Ahmad, I. Dynamic Critical Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, 1996.

[13] Kwok, Y.-K. and Ahmad, I. Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm. *Journal of Parallel and Distributed Computing*, 47:58–77, 1997.

[14] Kwok, Y.-K. and Ahmad, I. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[15] Malloy, B.A., Lloyd, E.L., and Soffa, M.L. Scheduling DAG's for Asynchronous Multiprocessor Execution. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):498–508, 1994.

[16] Manoharan, S. and Thanisch, P. Assigning Dependency Graphs onto Processor Networks. *Parallel Computing*, 17:63–73, 1991.

[17] Ribeiro, C.C. Test Instances for Scheduling Unrelated Processors under Precedence Constraints. Personal communication, 2002.

[18] Porto, S.C.S. and Ribeiro, C.C. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors Under Precedence Constraints. *International Journal of High-Speed Computing*, 7:47–71, 1995.

[19] Porto, S.C.S. and Ribeiro, C.C. Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling Under Precedence Constraints. *Journal of Heuristics*, 1(2):207–223, 1996.

[20] Porto, S.C.S., Kitajima, J.P.F.W., and Ribeiro, C.C. Performance Evaluation of a Parallel Tabu Search Task Scheduling Algorithm. *Parallel Computing*, 26:73–90, 2000.

[21] Sarje, A.K. and Sagar, G. Heuristic Model for Task Allocation in Distributed Computer Systems. *IEE Proceedings-E*, 138(5):313–318, 1991.

[22] Sih, G.C. and Lee, E.A. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, 1993.

[23] Tobita, T. and Kasahara, H. A Standard Task Graph Set for Fair Evaluation of Multiprocessor Scheduling algorithms. *Journal of Scheduling* 5(5), 379–394, 2002.

[24] Ullman, J.D. NP-Complete Scheduling Problems. *J. Comput. Syst. Sci.*, 10(3):384–393, 1975.

[25] Varvarigou, T.A., Roychowdhury, V.P., Kailath, T., and Lawler, E. Scheduling in and out Forests in the Presence of Communication Delays. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1065–1074, 1996.

Appendix

This Appendix includes the complete set of graphs representing the deviations from optimality of the solutions obtained by the four considered constructive heuristics for different task graph and multiprocessor architecture characteristics.

Figures 17 to 48 display the scheduling results obtained by using CPES, for all values of p and ρ. The deviations of the heuristic solutions from the optima, according to the value for ρ for all selected multiprocessor architectures, are given in Figures 20 to 29. Starting with Figure 30, scheduling results for a given ρ value and different multiprocessor system architecture are given in each figure. When scheduling independent tasks, communication delays and processor connection are not significant, therefore, there is only one graph displaying the deviations of the heuristic schedule from the optimal one.

The results for the LBMC, PPS, and DC heuristics are displayed starting with Figure 49.

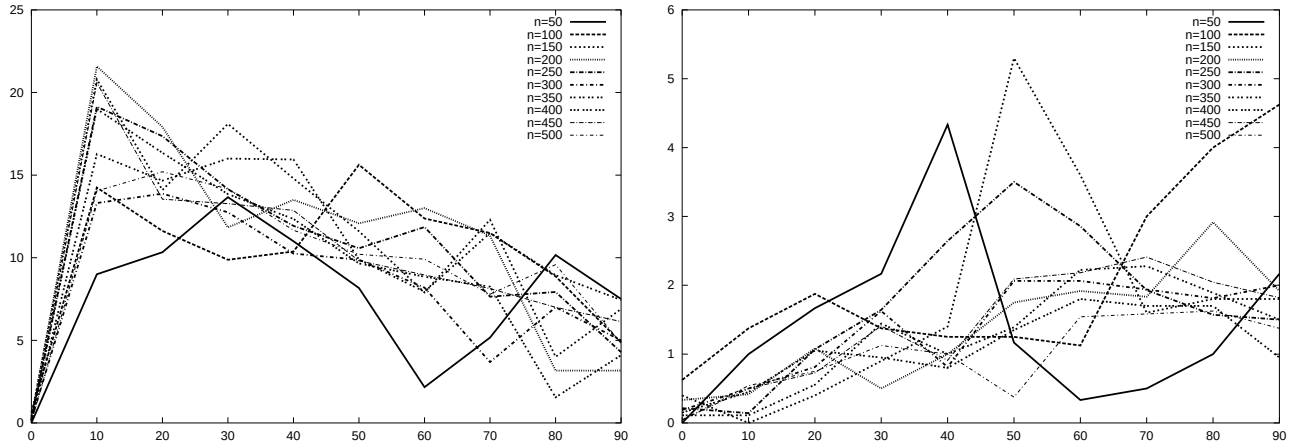

Figure 17: Solution deviations for 2-processor system, with and without communication delays

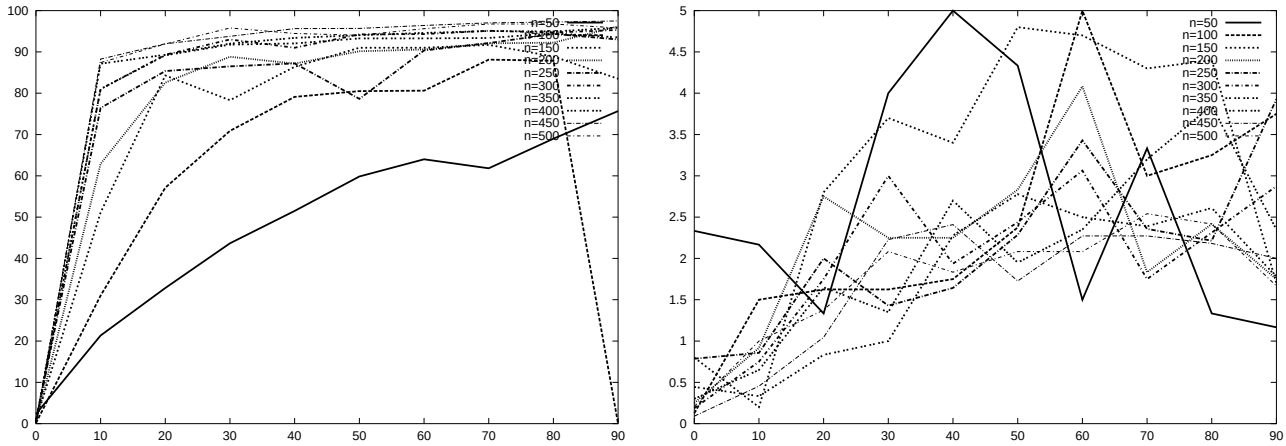

Figure 18: Solution deviations for 4-processor hypercube, with and without communication delays

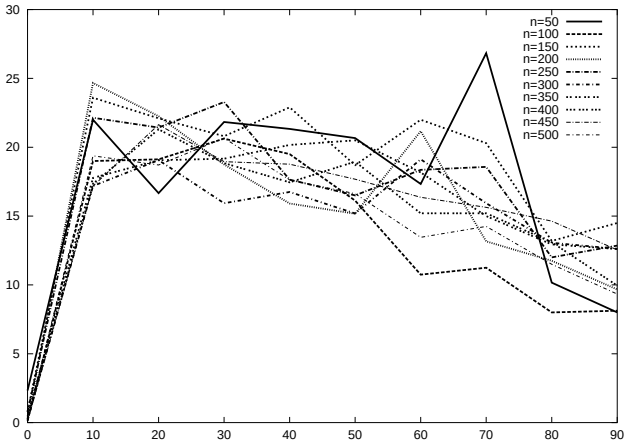

Figure 19: Solution deviations for completely connected 4 processors with communication delays

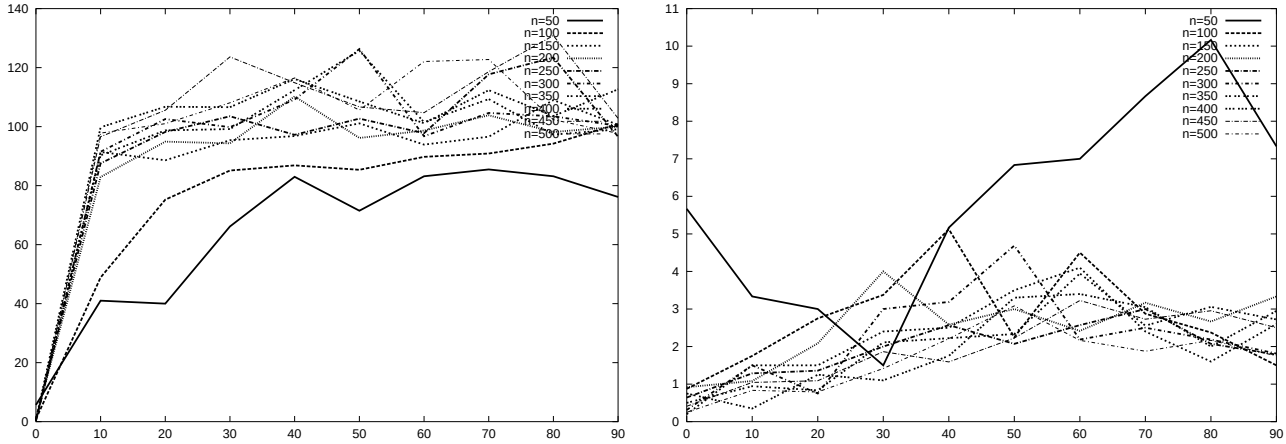

Figure 20: Solution deviations for mesh containing 6 processors, with and without communication delays

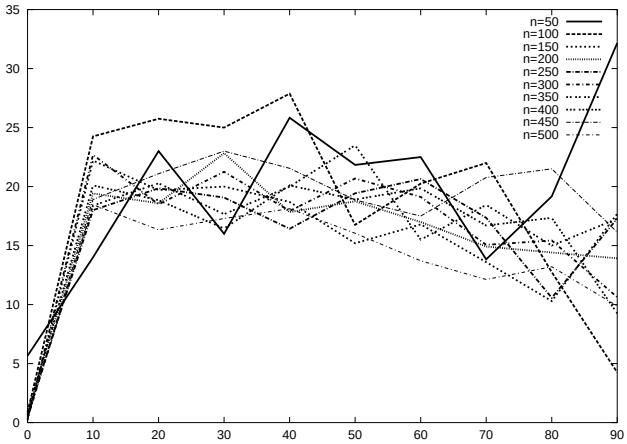

Figure 21: Solution deviations for completely connected 6 processors with communication delays

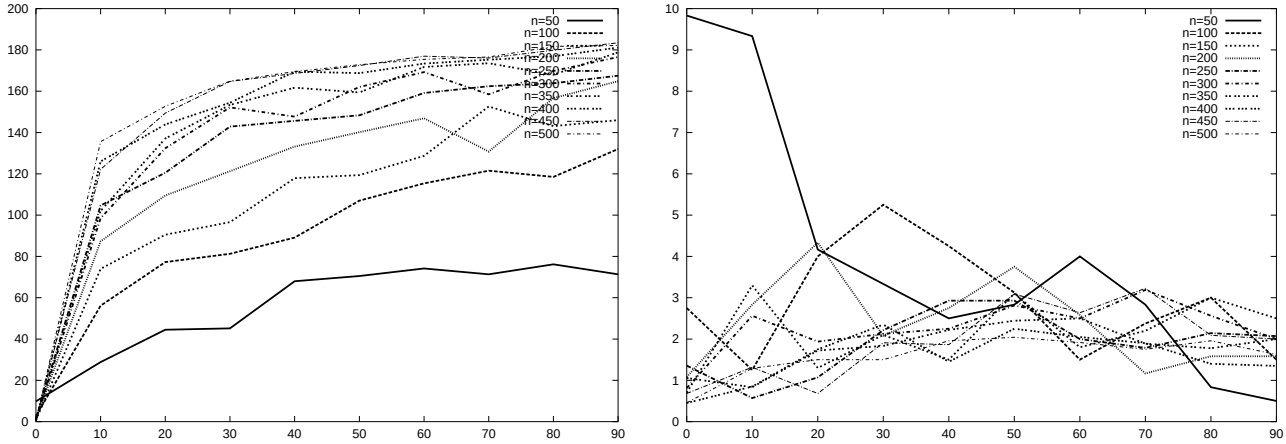

Figure 22: Solution deviations for 8-processor hypercube, with and without communication delays

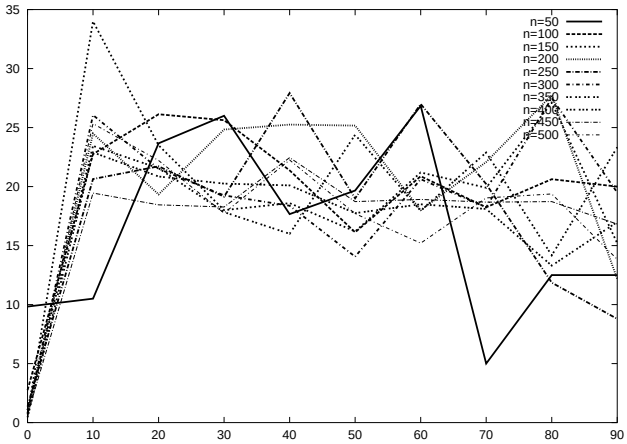

Figure 23: Solution deviations for completely connected 8 processors with communication delays

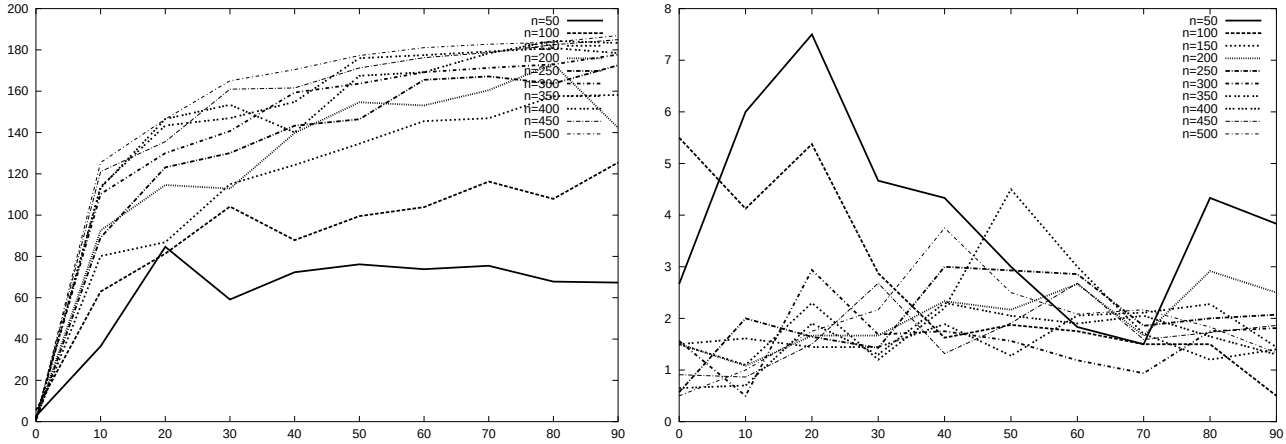

Figure 24: Solution deviations for 9-processor mesh, with and without communication delays

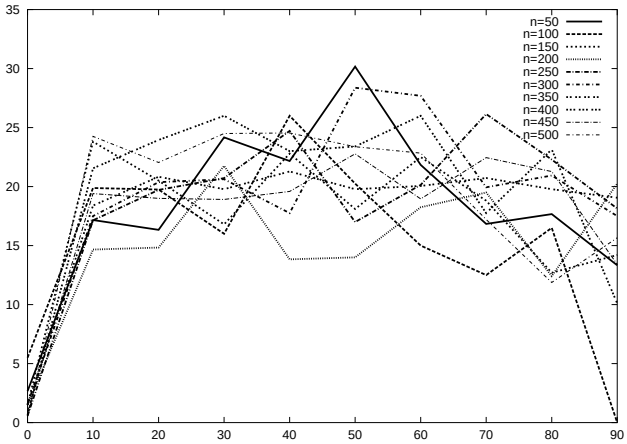

Figure 25: Solution deviations for completely connected 9 processors with communication delays

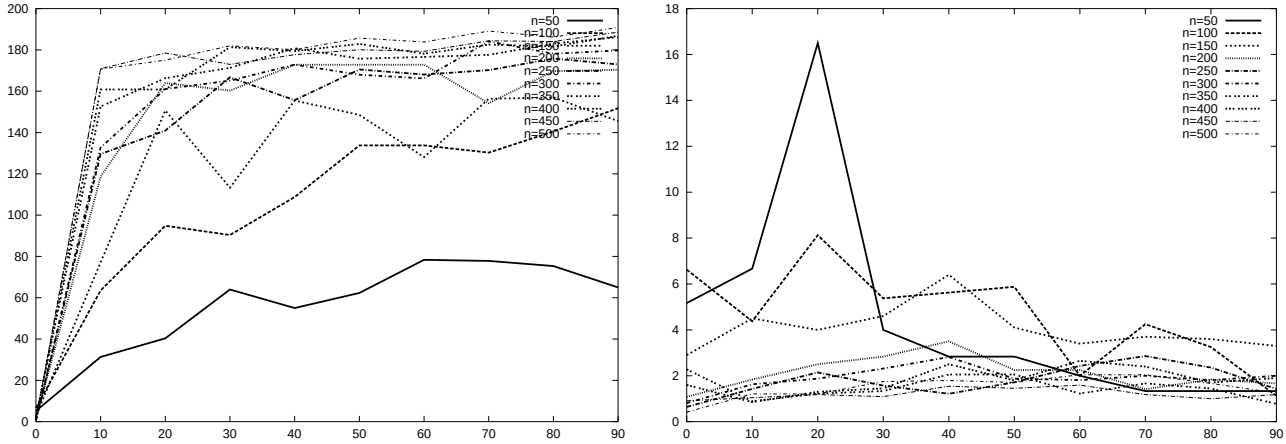

Figure 26: Solution deviations for 12-processor mesh, with and without communication delays

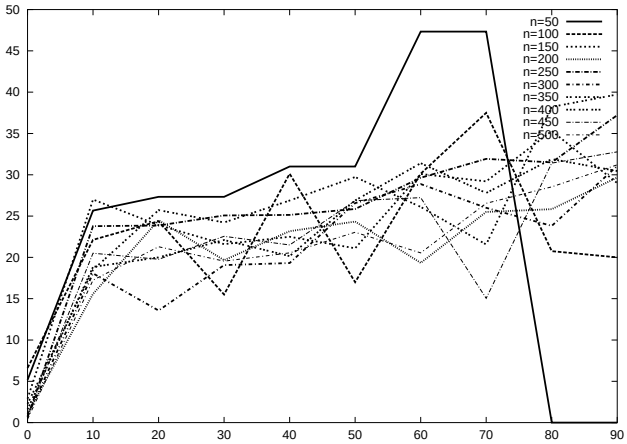

Figure 27: Solution deviations for completely connected 12 processors with communication delays

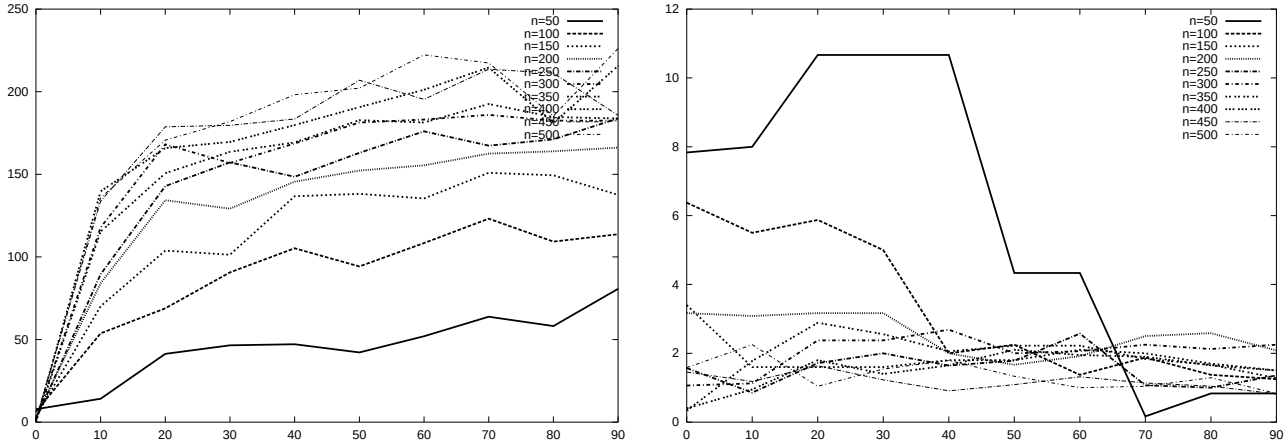

Figure 28: Solution deviations for 16-processor hypercube, with and without communication delays

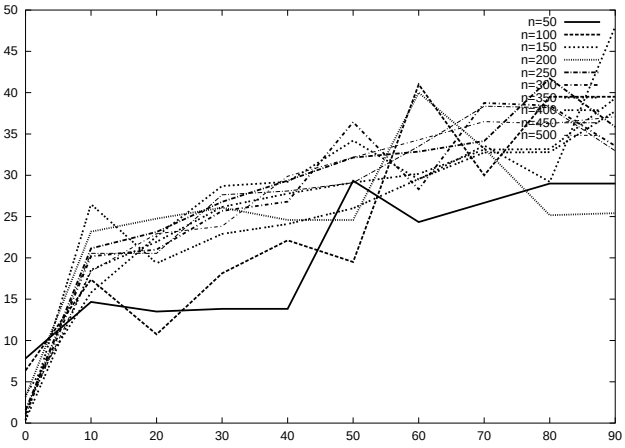

Figure 29: Solution deviations for completely connected 16 processors with communication delays

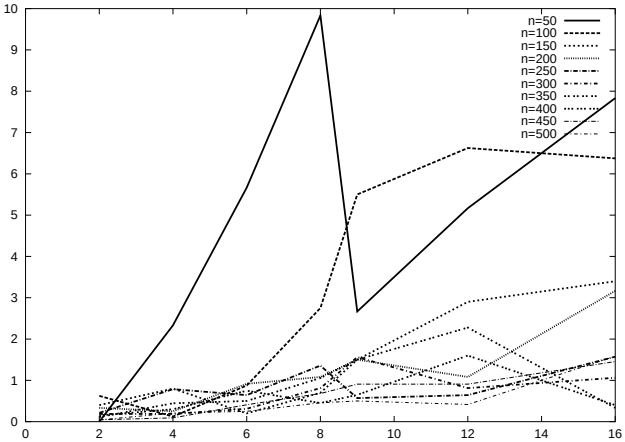

Figure 30: Solution deviations for scheduling independent tasks

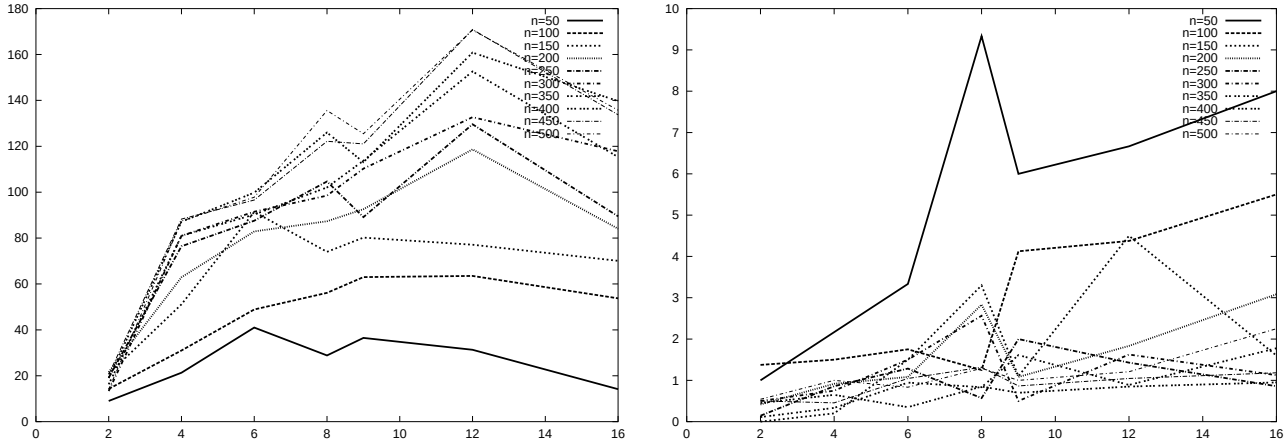

Figure 31: Solution deviations for sparse task graphs $\rho = 10$, with and without communication delays scheduled onto incomplete networks

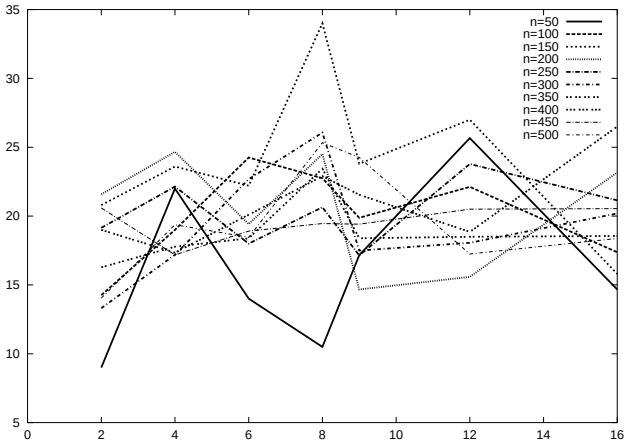

Figure 32: Solution deviations for sparse task graphs $\rho = 10$ with communication delays scheduled onto complete network of processors

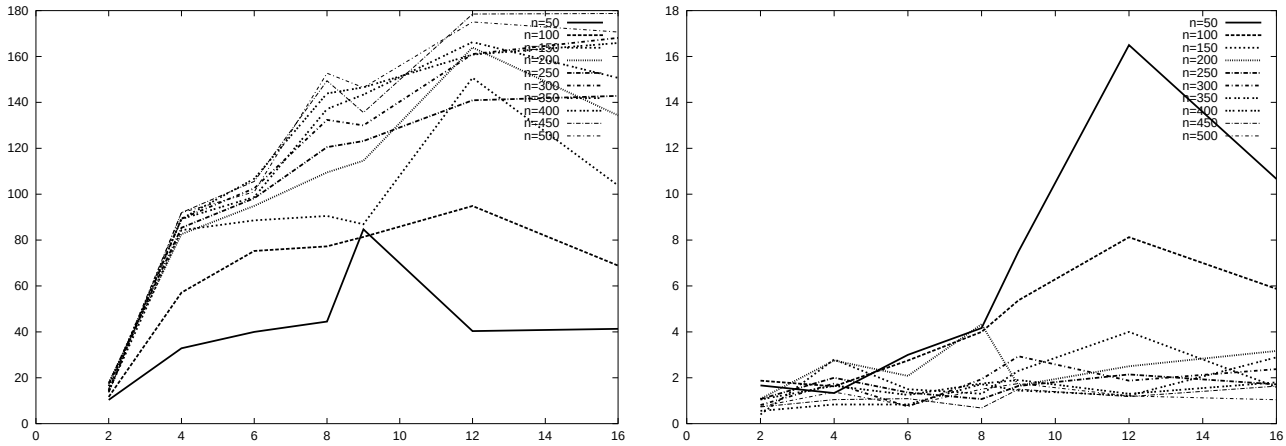

Figure 33: Solution deviations for task graphs with $\rho = 20$, with and without communication delays scheduled onto incomplete networks

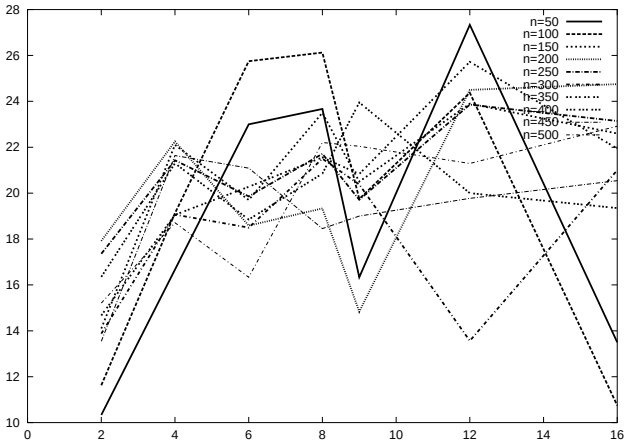

Figure 34: Solution deviations for task graphs with $\rho = 20$ with communication delays scheduled onto complete network of processors

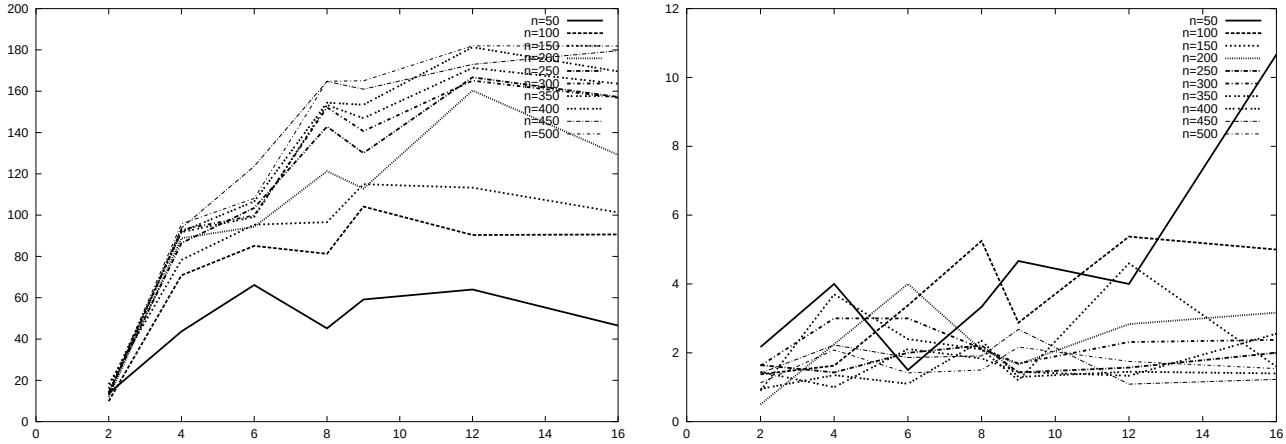

Figure 35: Solution deviations for task graphs with $\rho = 30$, with and without communication delays scheduled onto incomplete networks

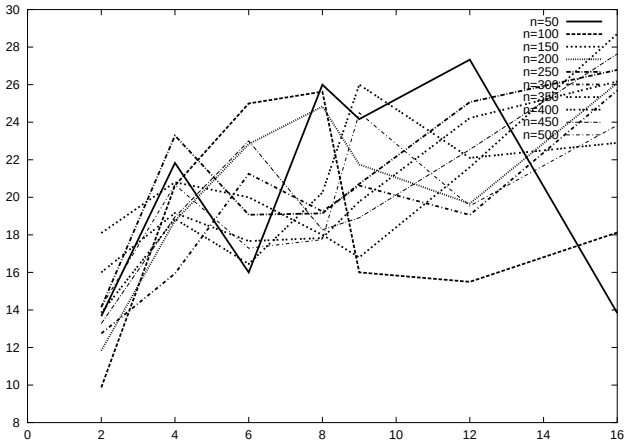

Figure 36: Solution deviations for task graphs with $\rho = 30$ with communication delays scheduled onto complete network of processors

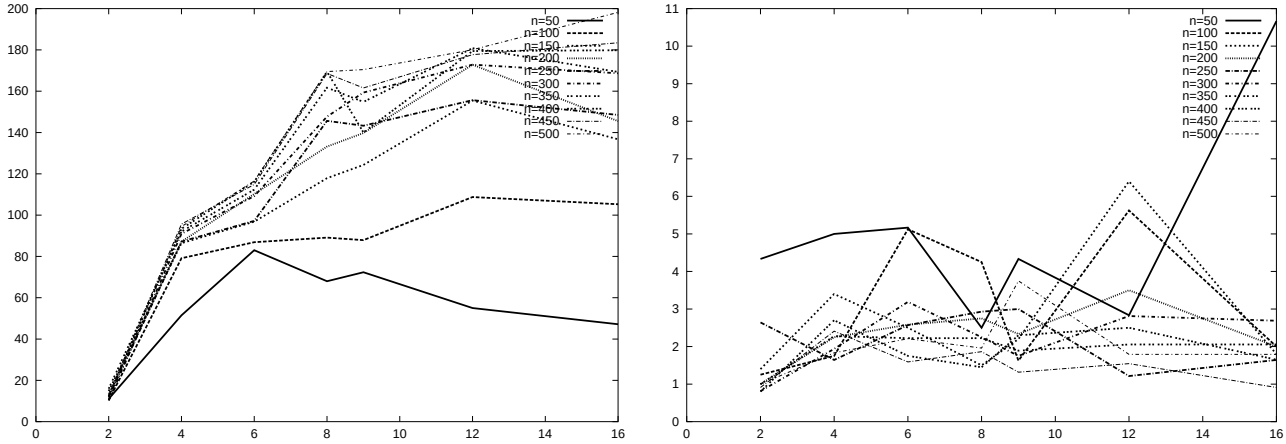

Figure 37: Solution deviations for task graphs with $\rho = 40$, with and without communication delays scheduled onto incomplete networks

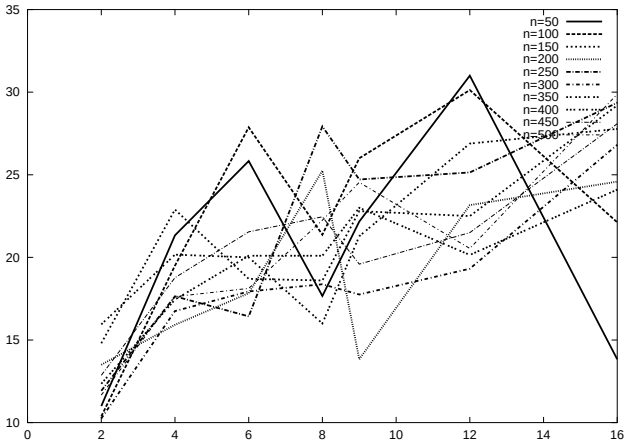

Figure 38: Solution deviations for task graphs with $\rho = 40$ with communication delays scheduled onto complete network of processors

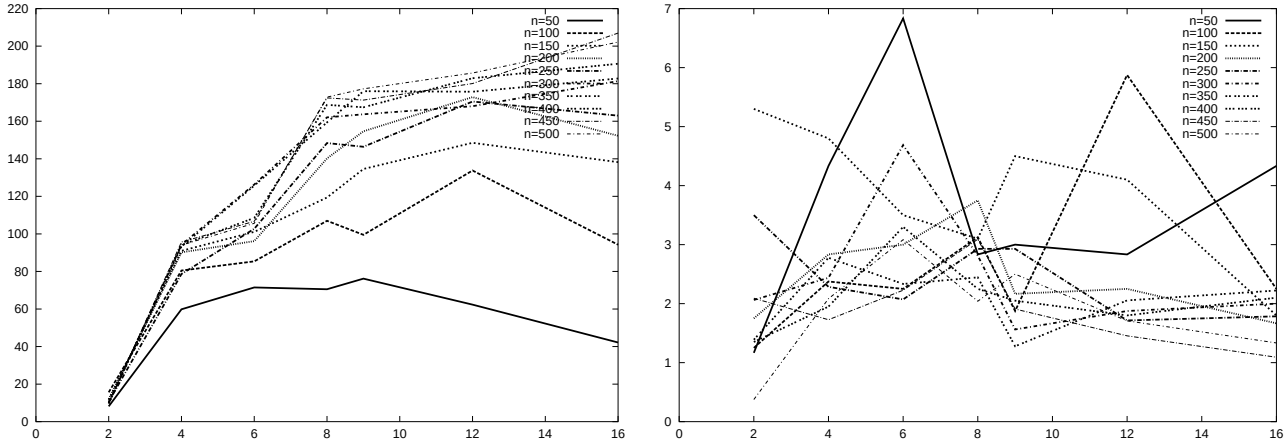

Figure 39: Solution deviations for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

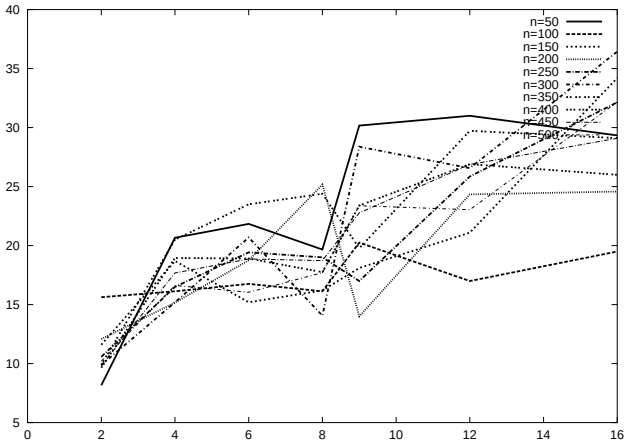

Figure 40: Solution deviations for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

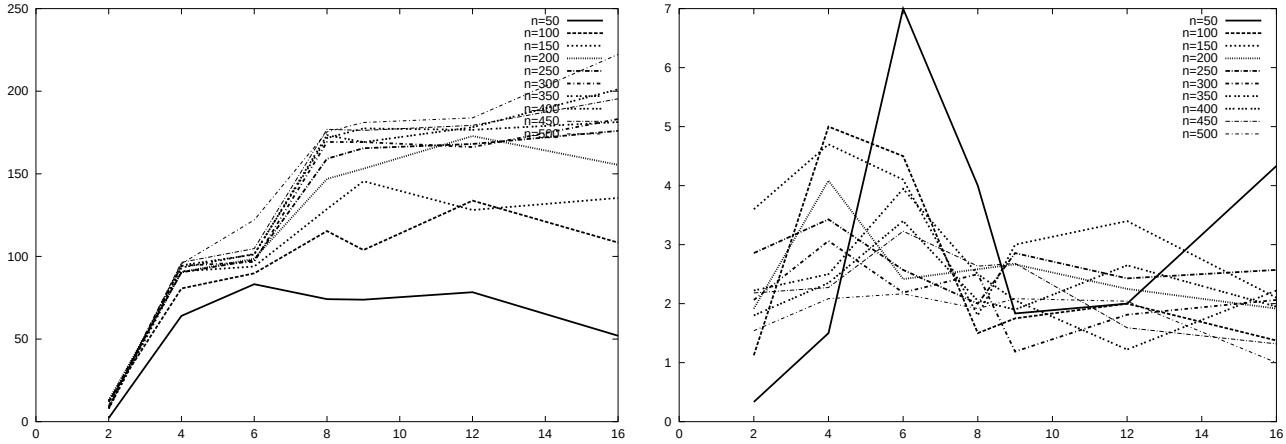

Figure 41: Solution deviations for task graphs with $\rho = 60$, with and without communication delays scheduled onto incomplete networks

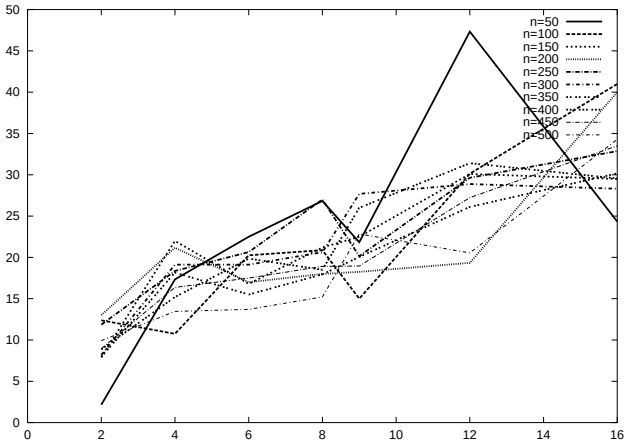

Figure 42: Solution deviations for task graphs with $\rho = 60$ with communication delays scheduled onto complete network of processors

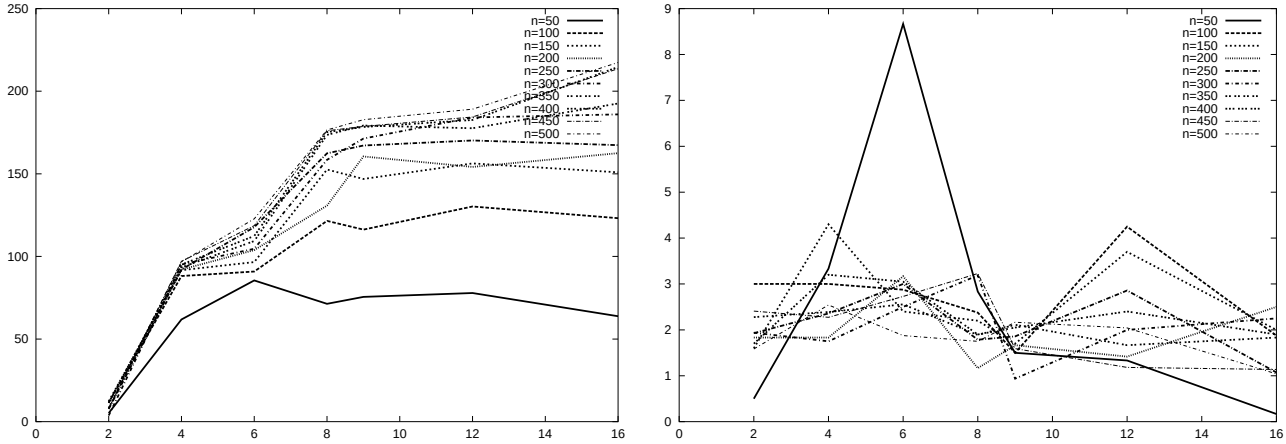

Figure 43: Solution deviations for task graphs with $\rho = 70$, with and without communication delays scheduled onto incomplete networks

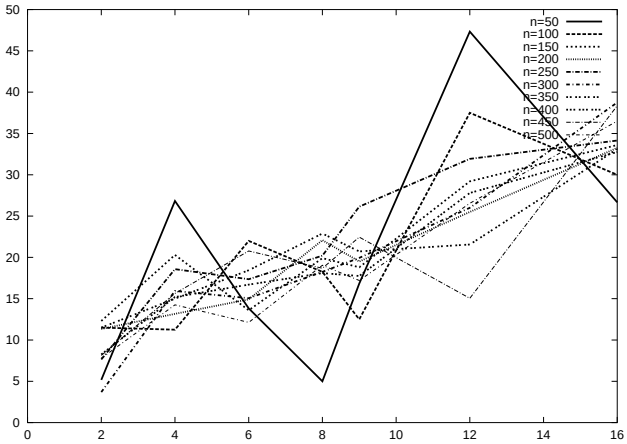

Figure 44: Solution deviations for task graphs with $\rho = 70$ with communication delays scheduled onto complete network of processors

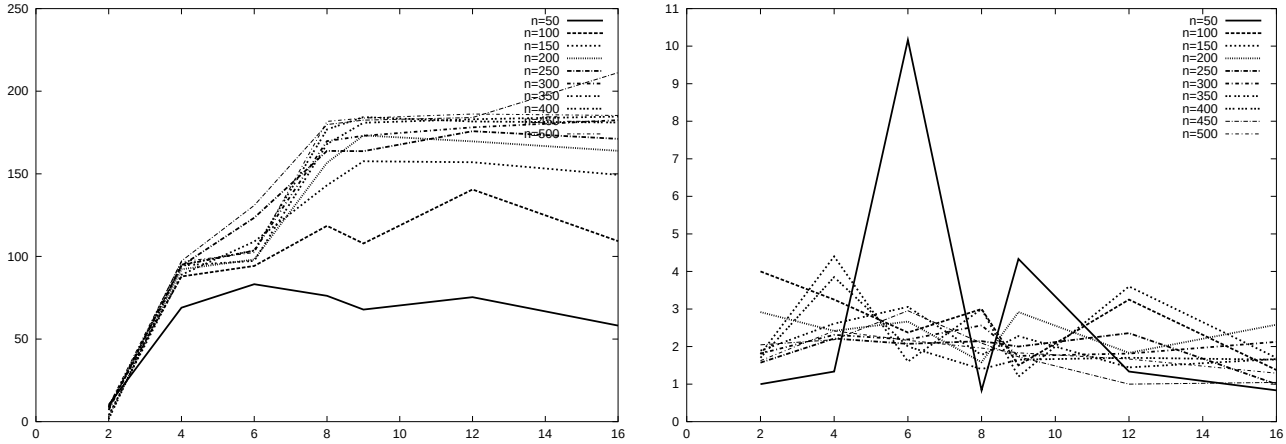

Figure 45: Solution deviations for task graphs with $\rho = 80$, with and without communication delays scheduled onto incomplete networks

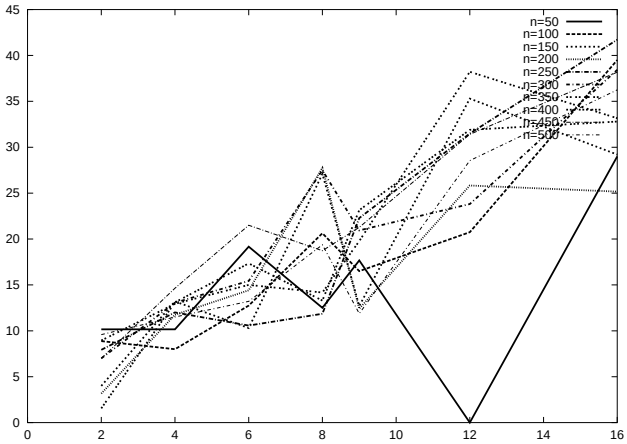

Figure 46: Solution deviations for task graphs with $\rho = 80$ with communication delays scheduled onto complete network of processors

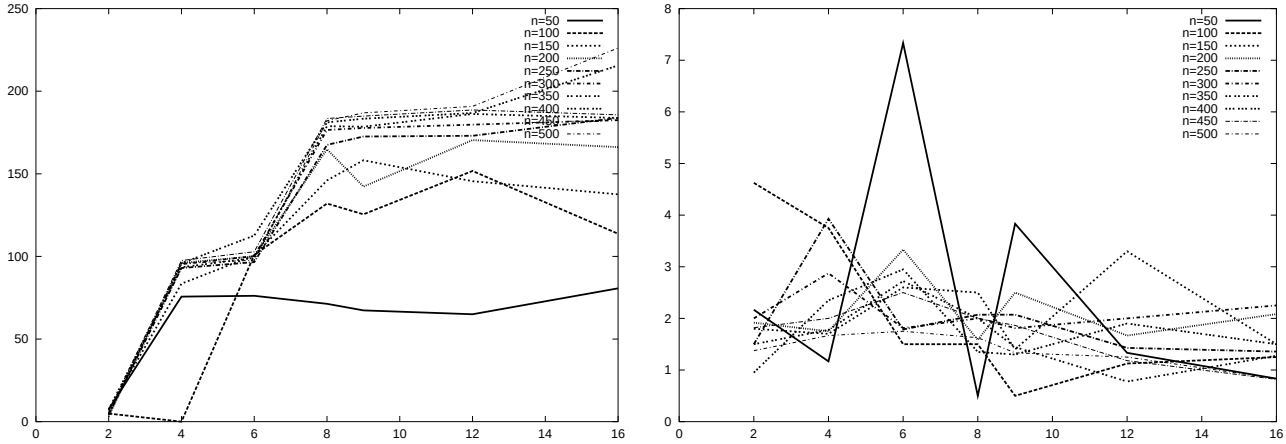

Figure 47: Solution deviations for task graphs with $\rho = 90$, with and without communication delays scheduled onto incomplete networks

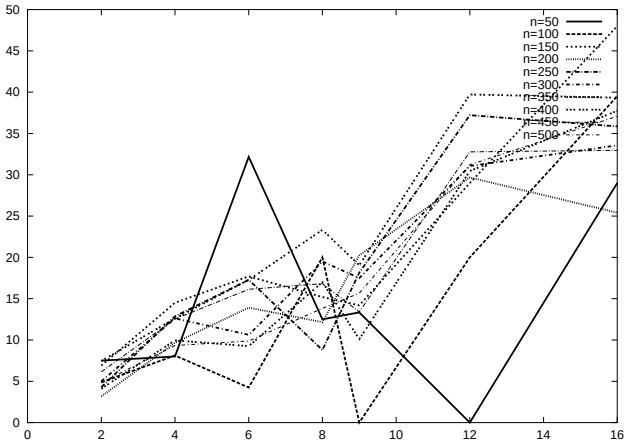

Figure 48: Solution deviations for task graphs with $\rho = 90$ with communication delays scheduled onto complete network of processors

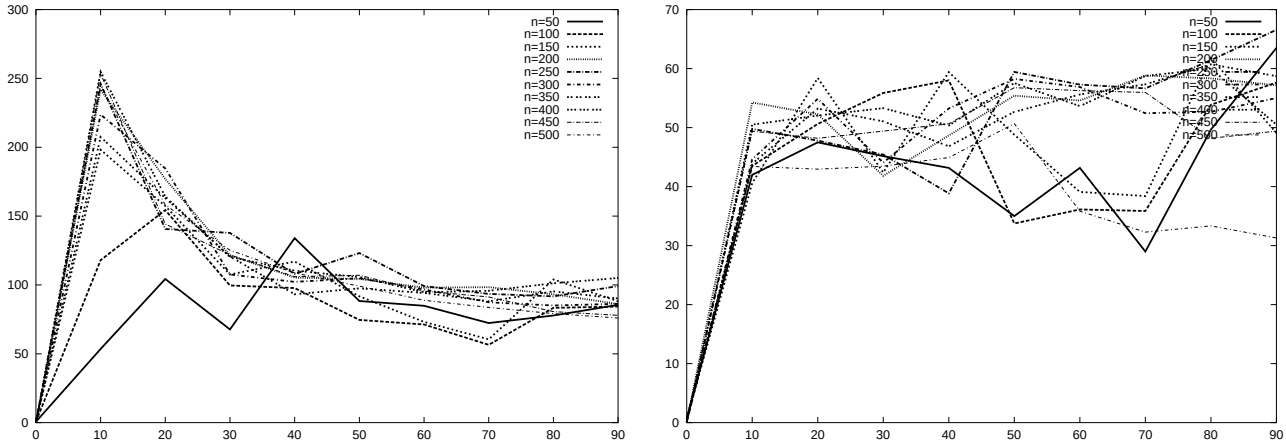

Figure 49: LBMC Solution deviations for 2-processor system, with and without communication delays

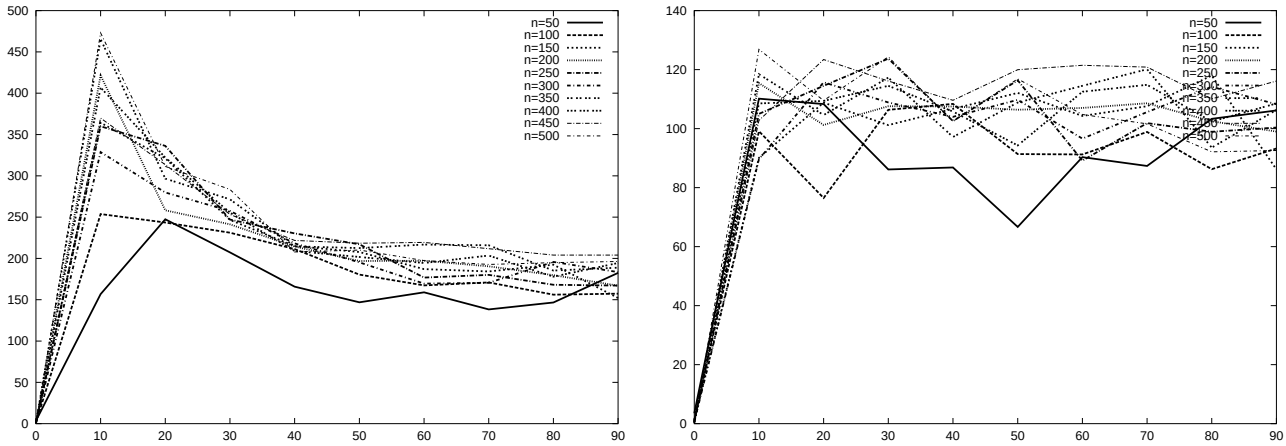

Figure 50: LBMC Solution deviations for 4-processor hypercube, with and without communication delays

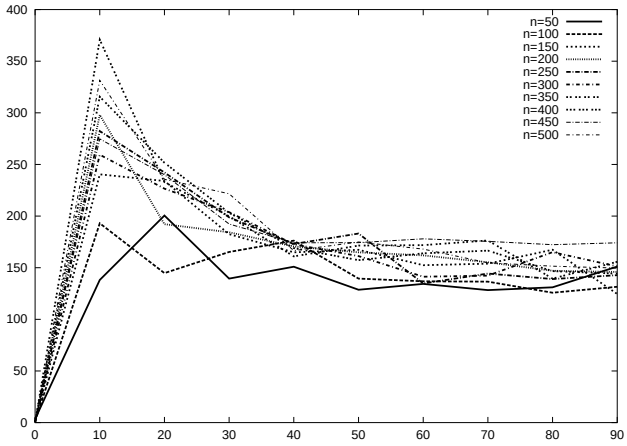

Figure 51: LBMC Solution deviations for completely connected 4 processors with communication delays

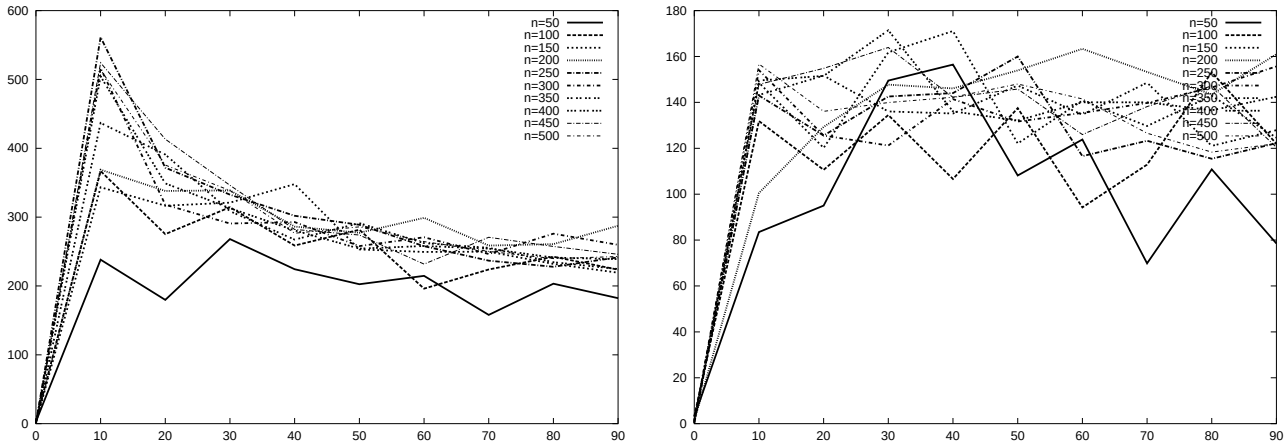

Figure 52: LBMC Solution deviations for 6-processor mesh, with and without communication delays

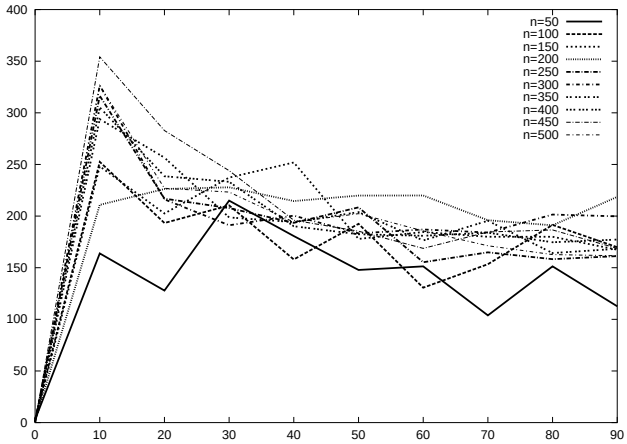

Figure 53: LBMC Solution deviations for completely connected 6 processors with communication delays

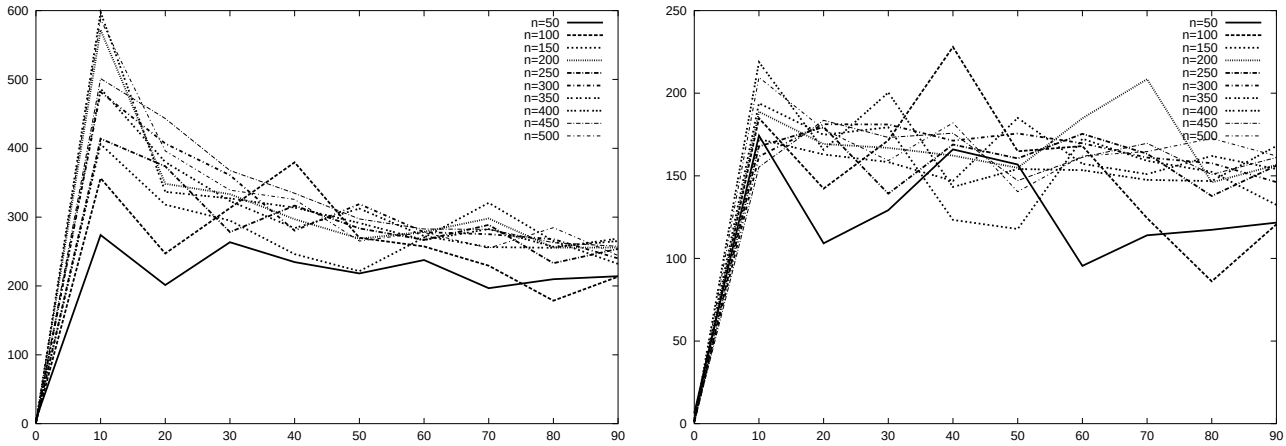

Figure 54: LBMC Solution deviations for 8-processor hypercube, with and without communication delays

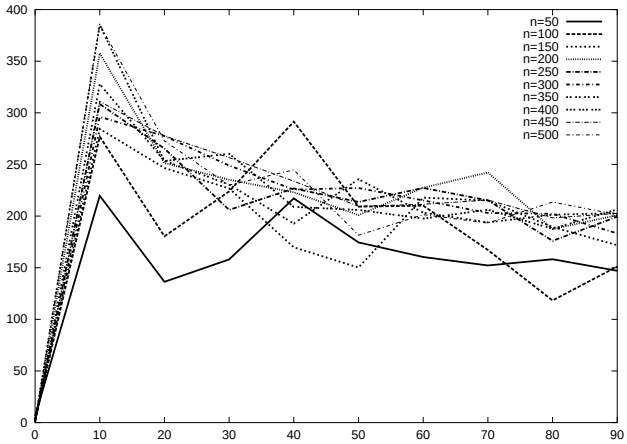

Figure 55: LBMC Solution deviations for completely connected 8 processors with communication delays

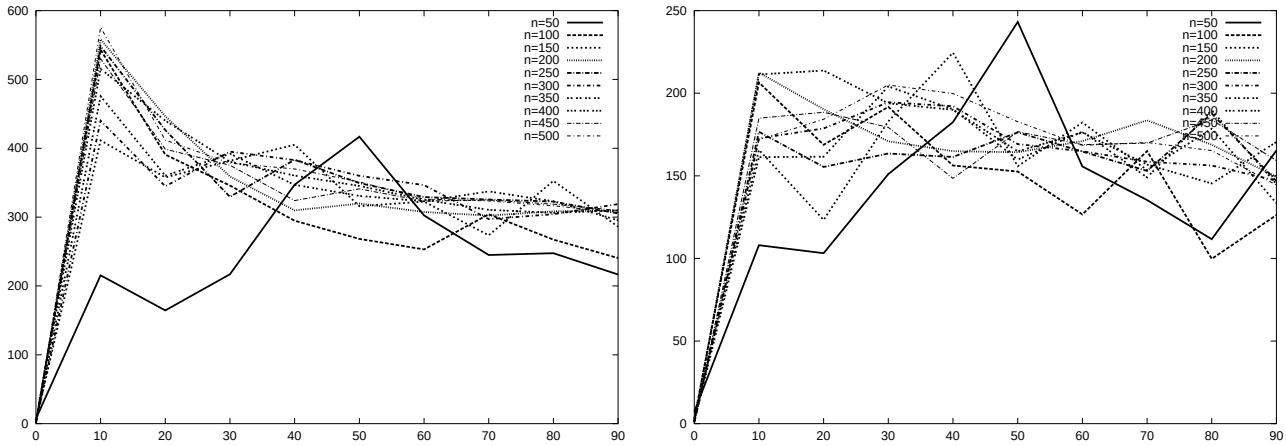

Figure 56: LBMC Solution deviations for 9-processor mesh, with and without communication delays

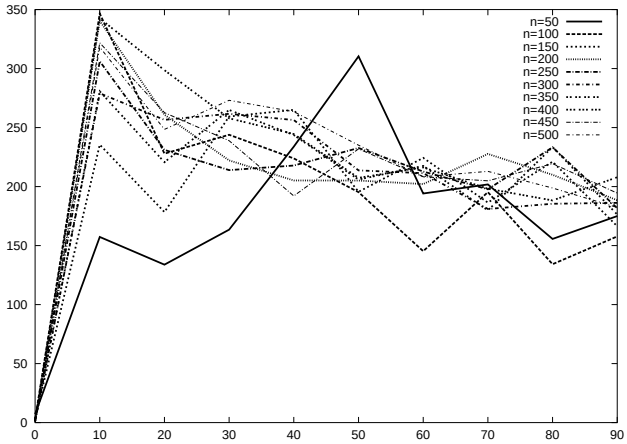

Figure 57: LBMC Solution deviations for completely connected 9 processors with communication delays

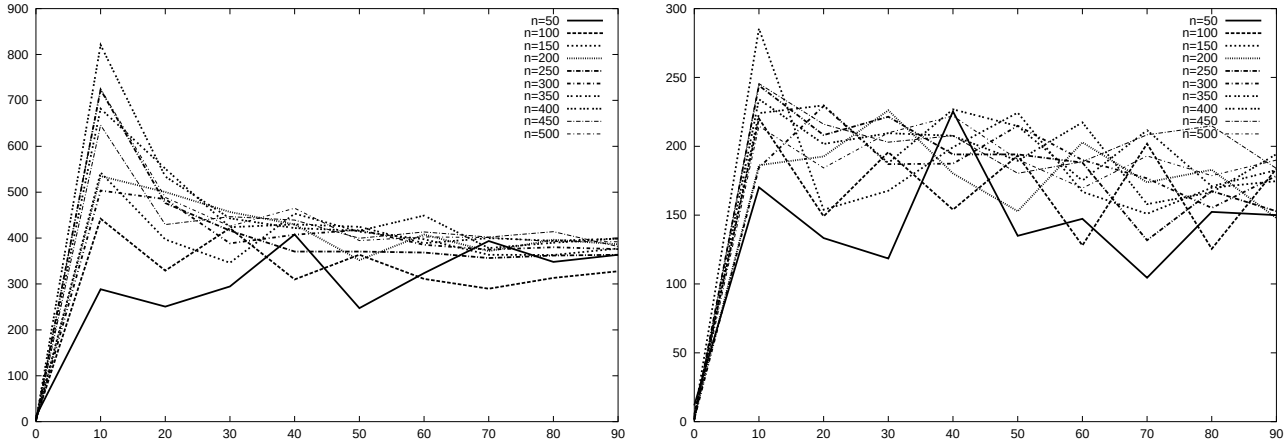

Figure 58: LBMC Solution deviations for 12-processor mesh, with and without communication delays

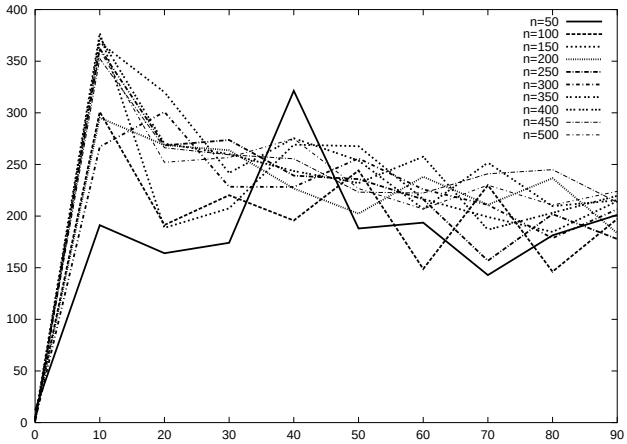

Figure 59: LBMC Solution deviations for completely connected 12 processors with communication delays

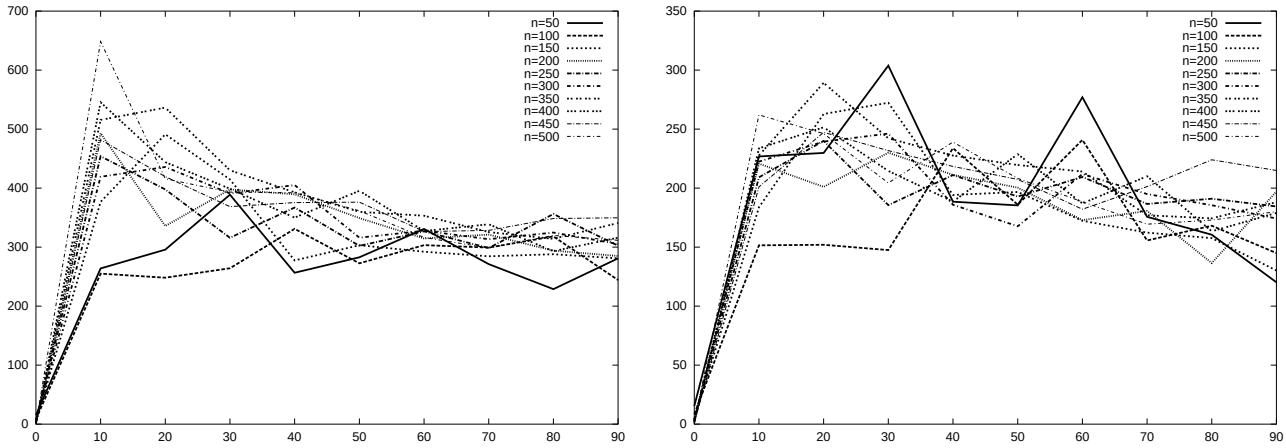

Figure 60: LBMC solution deviations for 16-processor hypercube, with and without communication delays

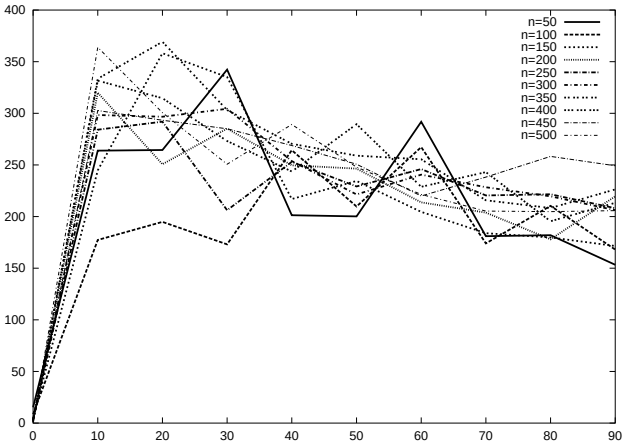

Figure 61: LBMC solution deviations for completely connected 16 processors with communication delays

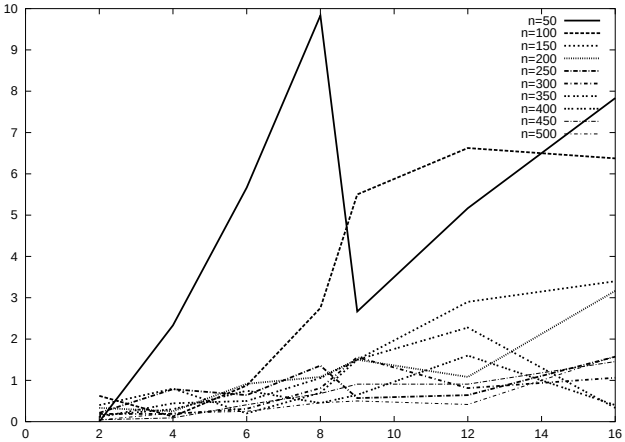

Figure 62: Solution deviations for scheduling independent tasks

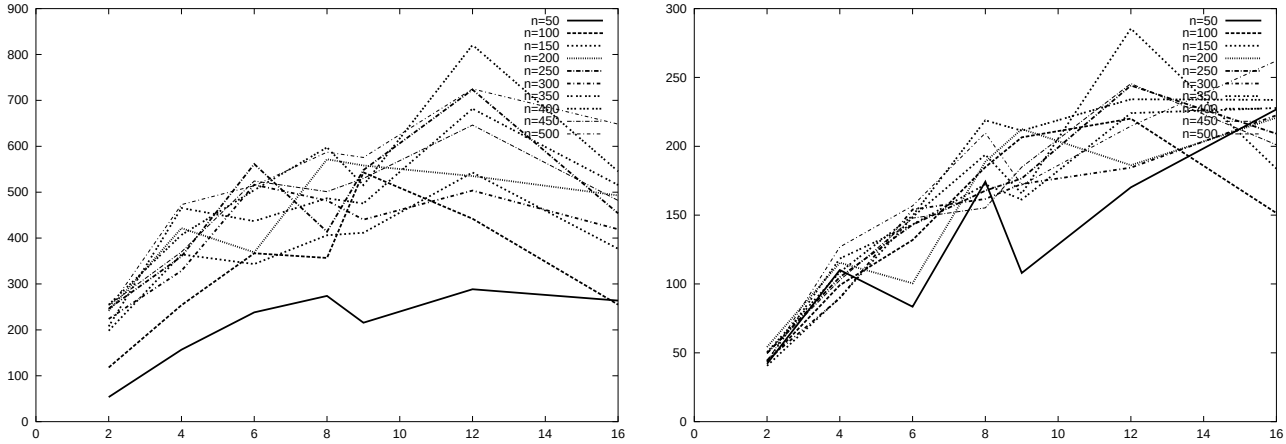

Figure 63: Solution deviations for sparse task graphs $\rho = 10$, with and without communication delays scheduled onto incomplete networks

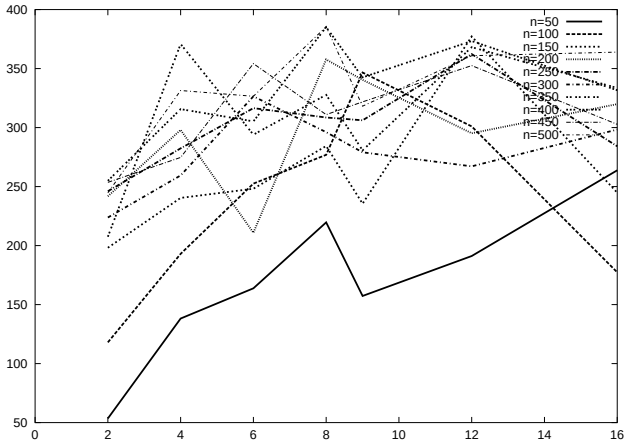

Figure 64: Solution deviations for sparse task graphs $\rho = 10$ with communication delays scheduled onto complete network of processors

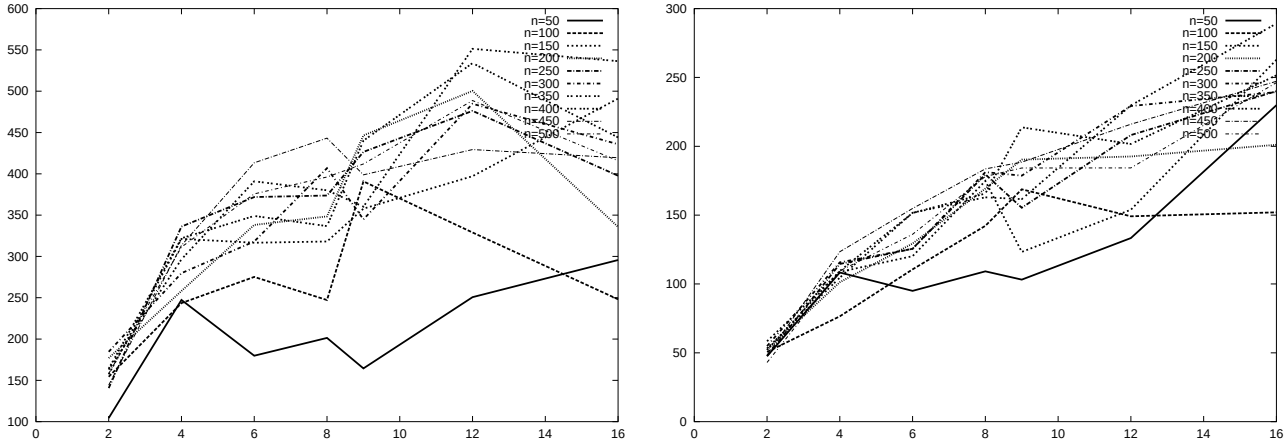

Figure 65: Solution deviations for task graphs with $\rho = 20$, with and without communication delays scheduled onto incomplete networks

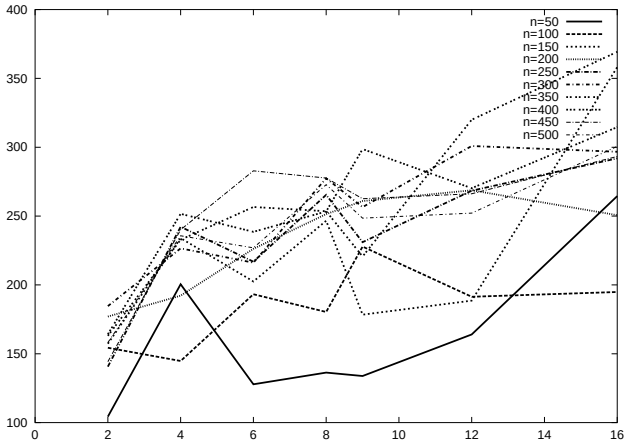

Figure 66: Solution deviations for task graphs with $\rho = 20$ with communication delays scheduled onto complete network of processors

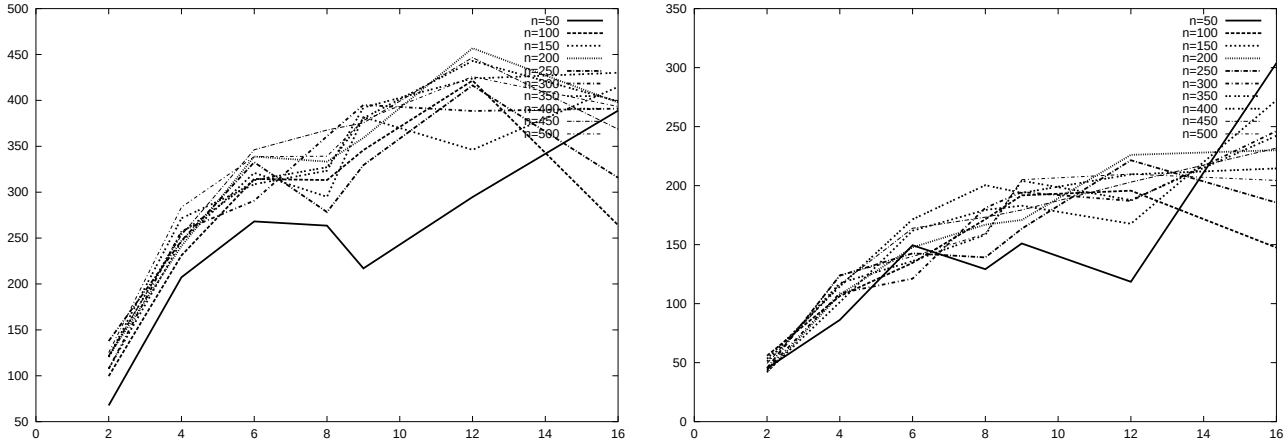

Figure 67: Solution deviations for task graphs with $\rho = 30$, with and without communication delays scheduled onto incomplete networks

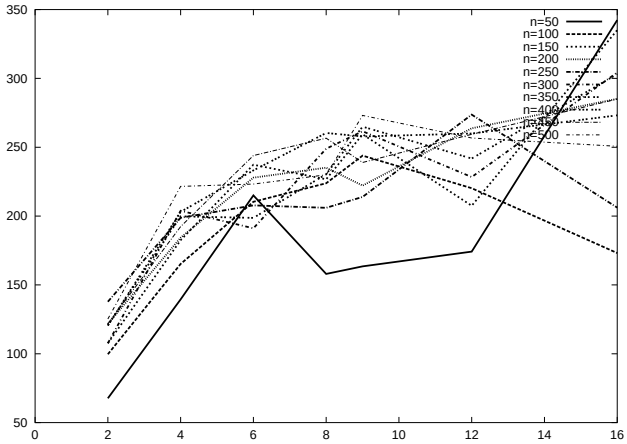

Figure 68: Solution deviations for task graphs with $\rho = 30$ with communication delays scheduled onto complete network of processors

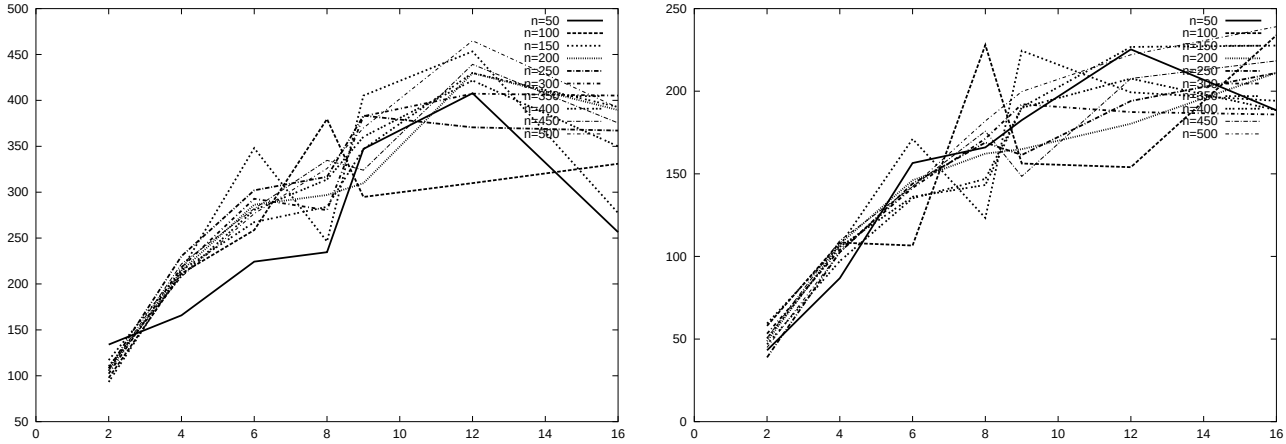

Figure 69: Solution deviations for task graphs with $\rho = 40$, with and without communication delays scheduled onto incomplete networks

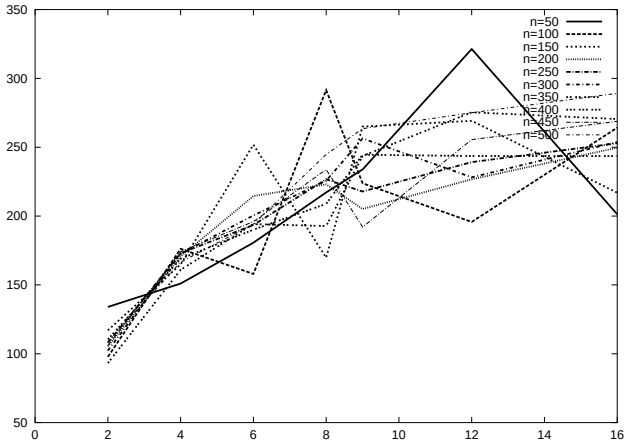

Figure 70: Solution deviations for task graphs with $\rho = 40$ with communication delays scheduled onto complete network of processors

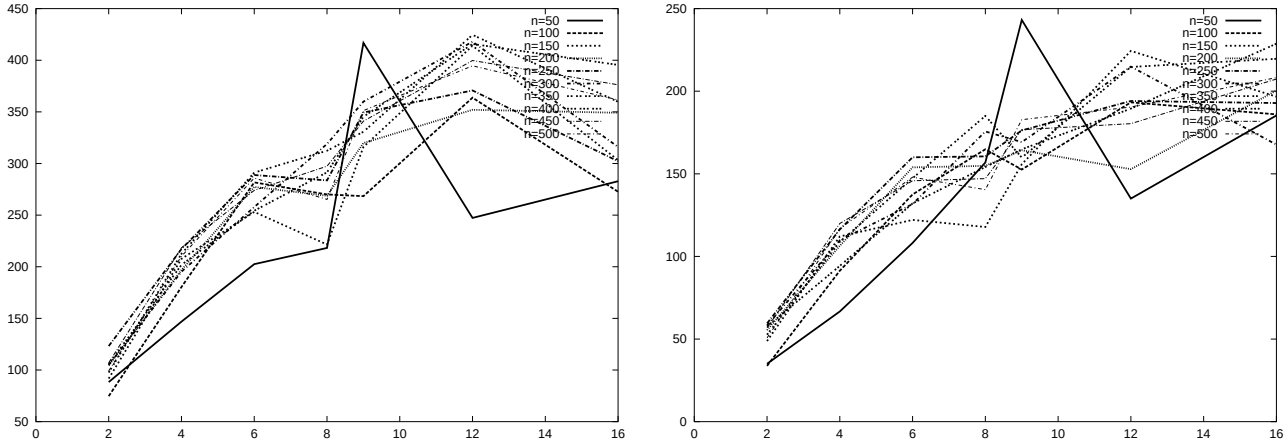

Figure 71: Solution deviations for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

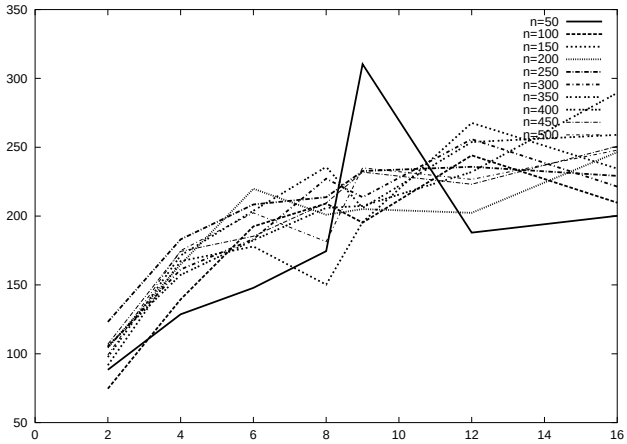

Figure 72: Solution deviations for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

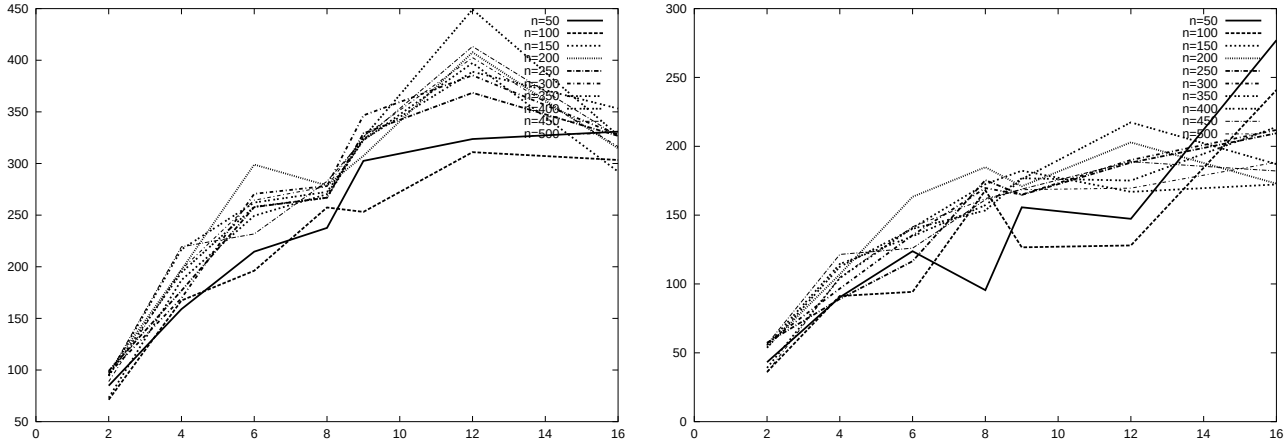

Figure 73: Solution deviations for task graphs with $\rho = 60$, with and without communication delays scheduled onto incomplete networks

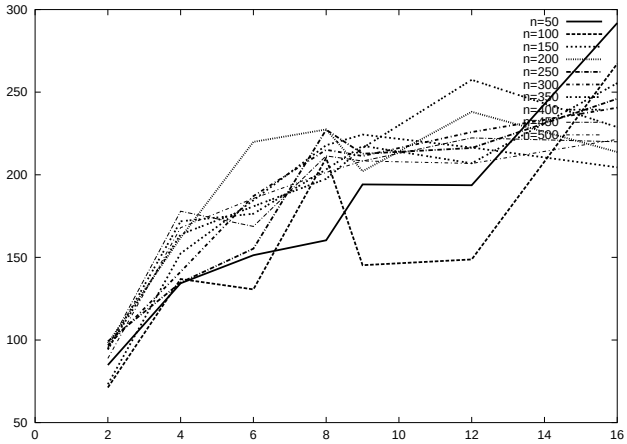

Figure 74: Solution deviations for task graphs with $\rho = 60$ with communication delays scheduled onto complete network of processors

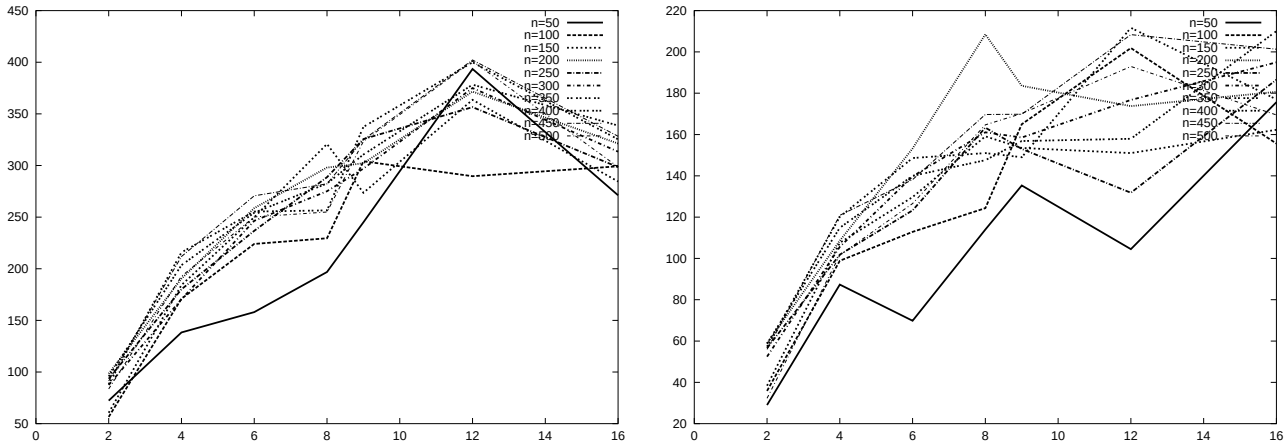

Figure 75: Solution deviations for task graphs with $\rho = 70$, with and without communication delays scheduled onto incomplete networks

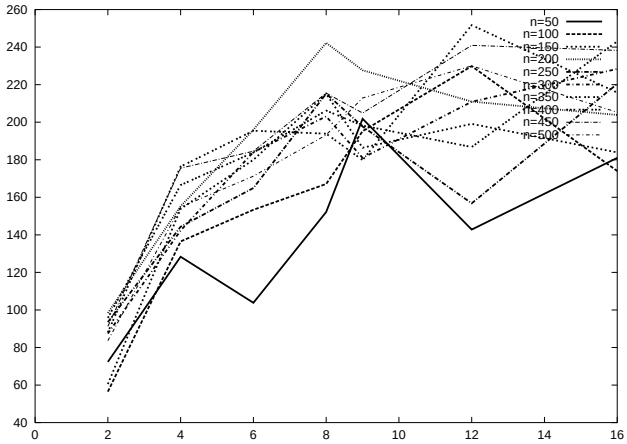

Figure 76: Solution deviations for task graphs with $\rho = 70$ with communication delays scheduled onto complete network of processors

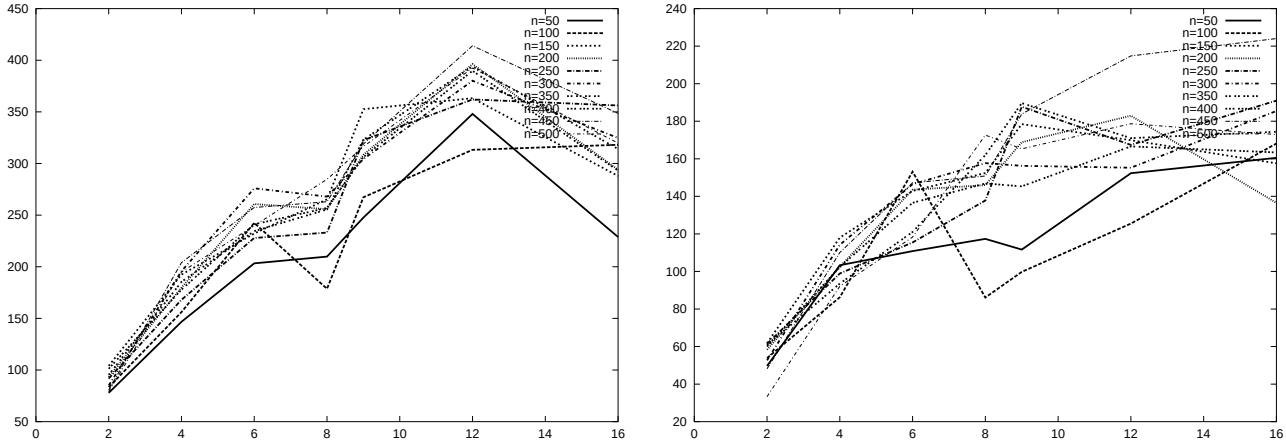

Figure 77: Solution deviations for task graphs with $\rho = 80$, with and without communication delays scheduled onto incomplete networks

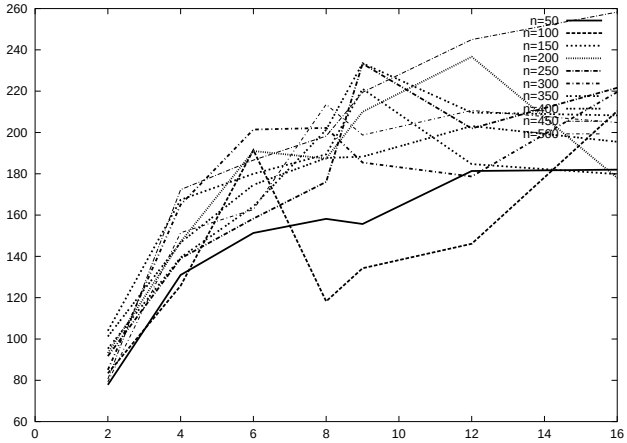

Figure 78: Solution deviations for task graphs with $\rho = 80$ with communication delays scheduled onto complete network of processors

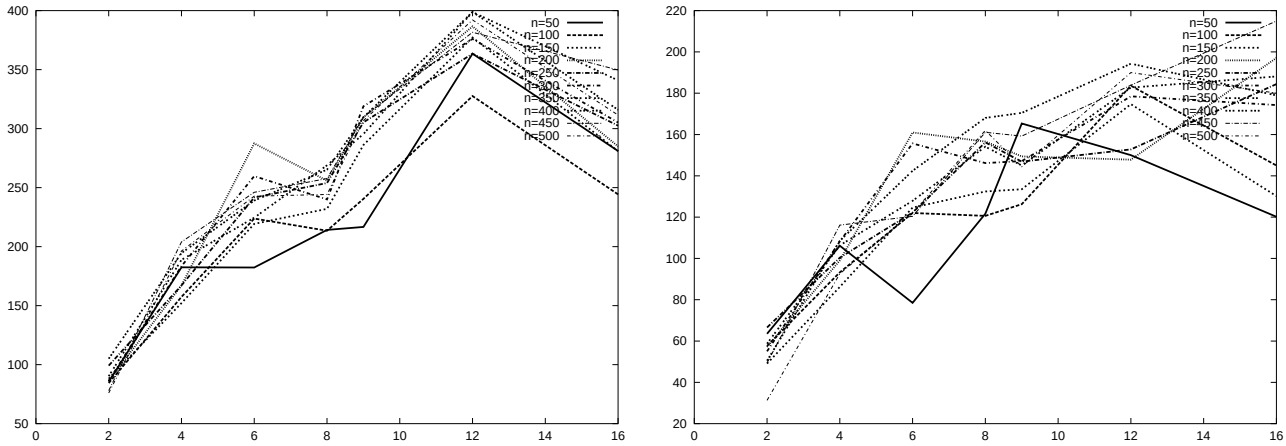

Figure 79: Solution deviations for task graphs with $\rho = 90$, with and without communication delays scheduled onto incomplete networks

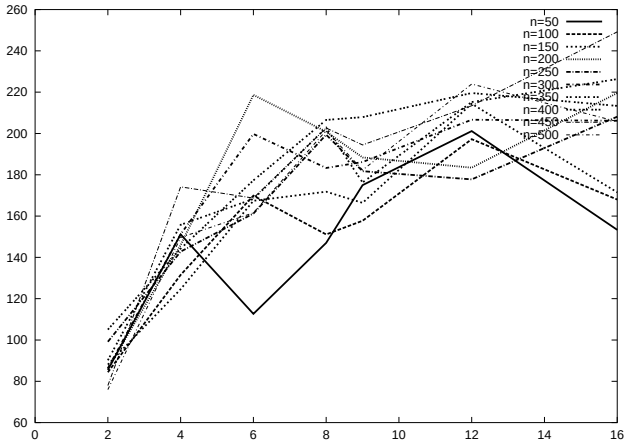

Figure 80: Solution deviations for task graphs with $\rho = 90$ with communication delays scheduled onto complete network of processors

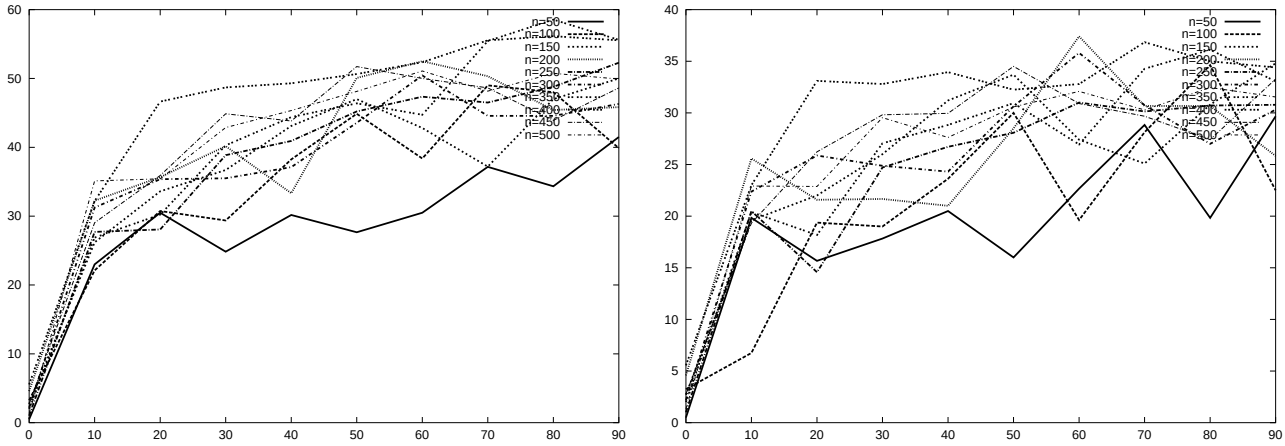

Figure 81: PPS solution deviations for 2-processor system, with and without communication delays

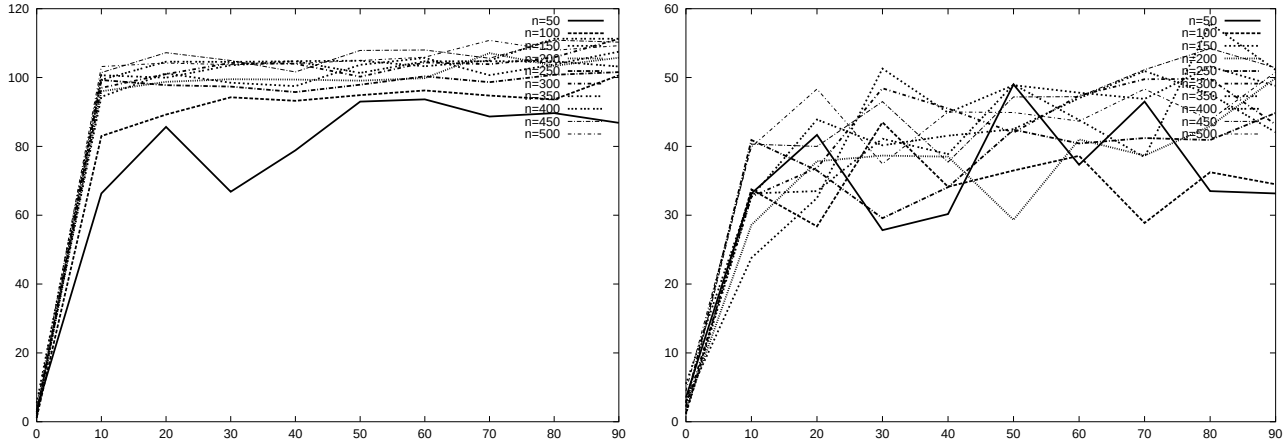

Figure 82: PPS solution deviations for 4-processor hypercube, with and without communication delays

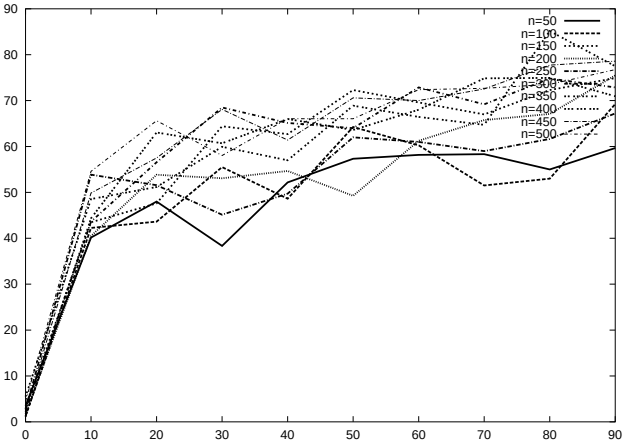

Figure 83: PPS solution deviations for completely connected 4 processors with communication delays

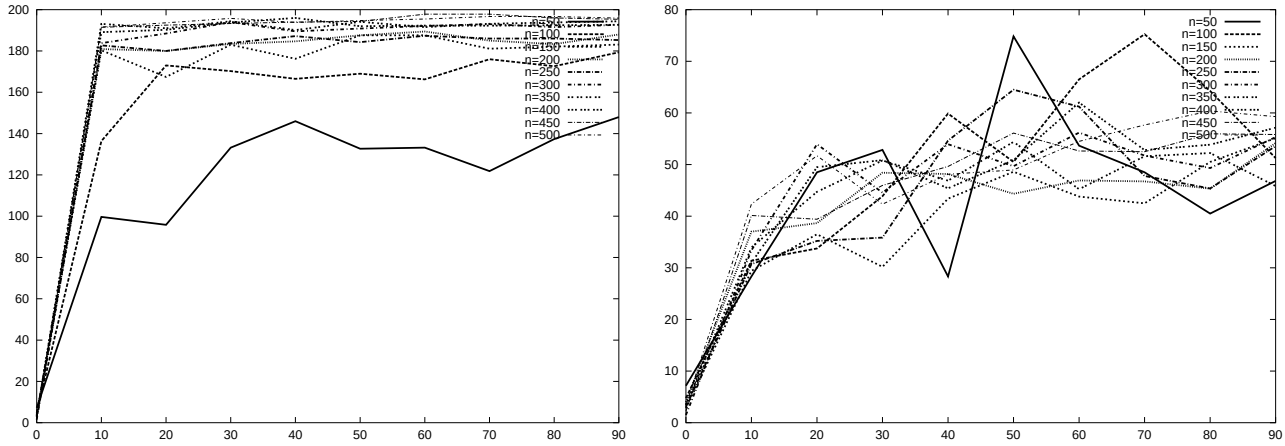

Figure 84: PPS solution deviations for 6-processor mesh, with and without communication delays

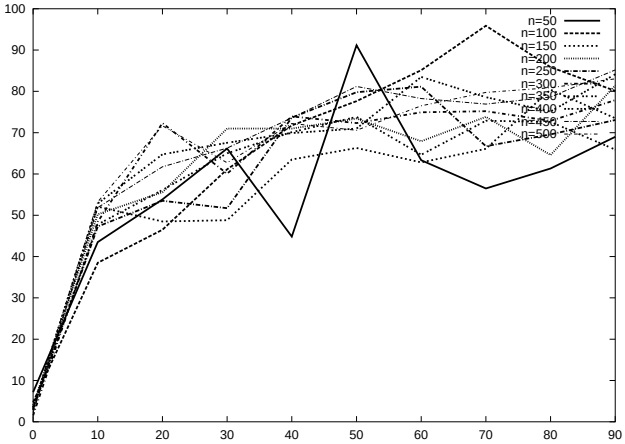

Figure 85: PPS solution deviations for completely connected 6 processors with communication delays

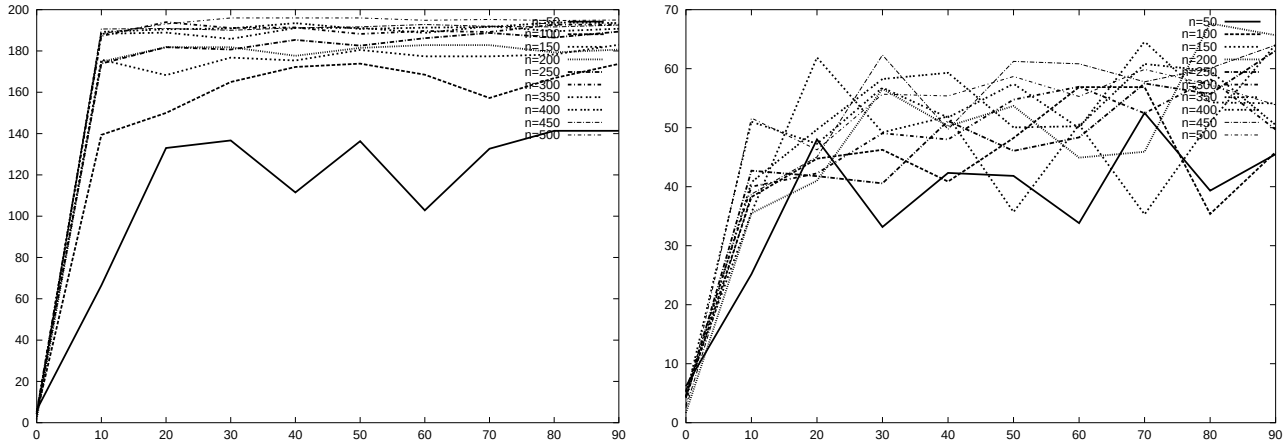

Figure 86: PPS solution deviations for 8-processor hypercube, with and without communication delays

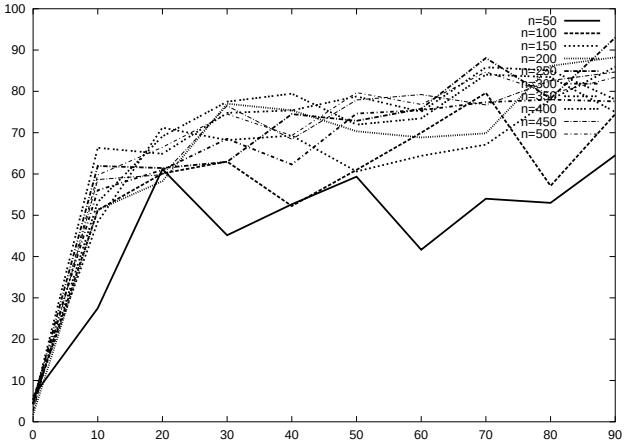

Figure 87: PPS solution deviations for completely connected 8 processors with communication delays

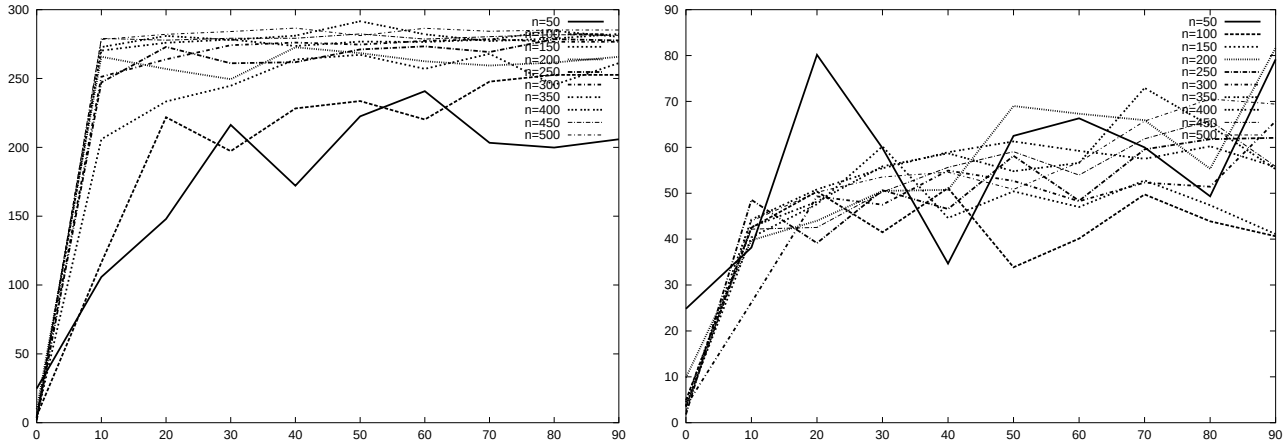

Figure 88: PPS solution deviations for 9-processor mesh, with and without communication delays

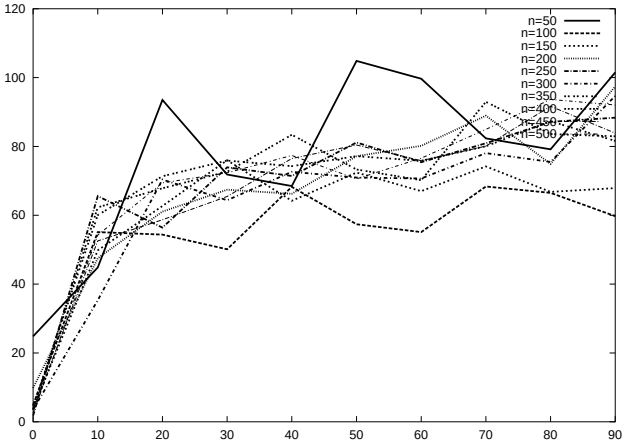

Figure 89: PPS solution deviations for completely connected 9 processors with communication delays

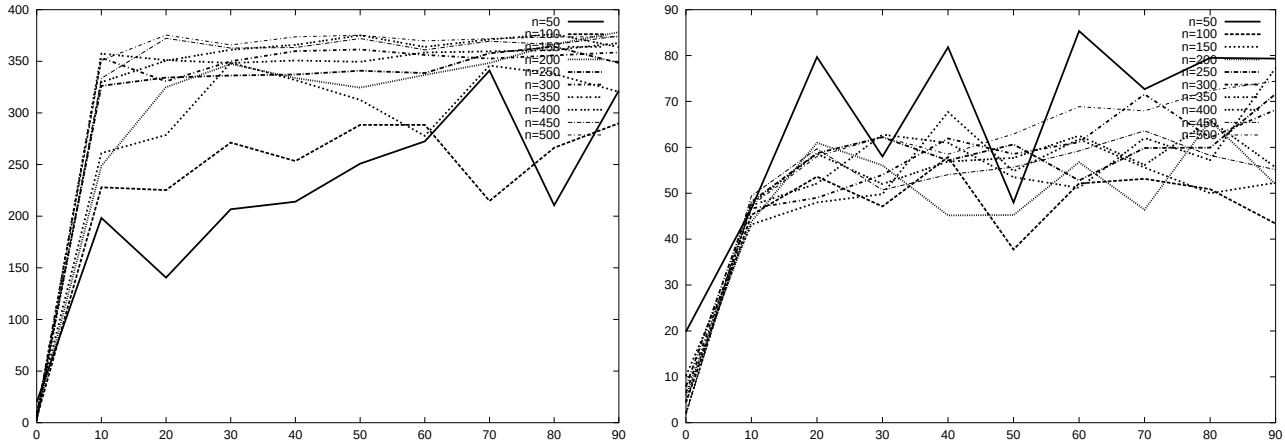

Figure 90: PPS solution deviations for 12-processor mesh, with and without communication delays

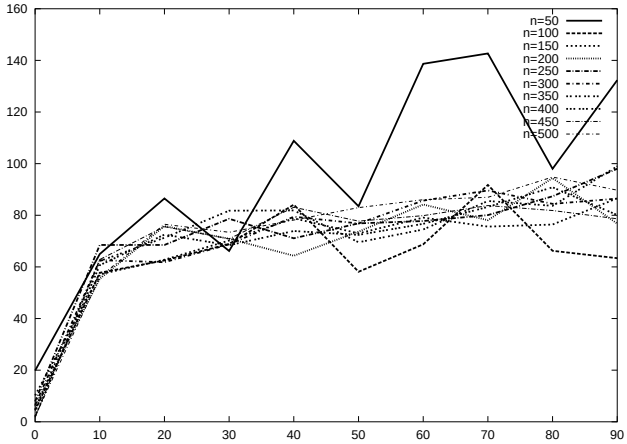

Figure 91: PPS solution deviations for completely connected 12 processors with communication delays

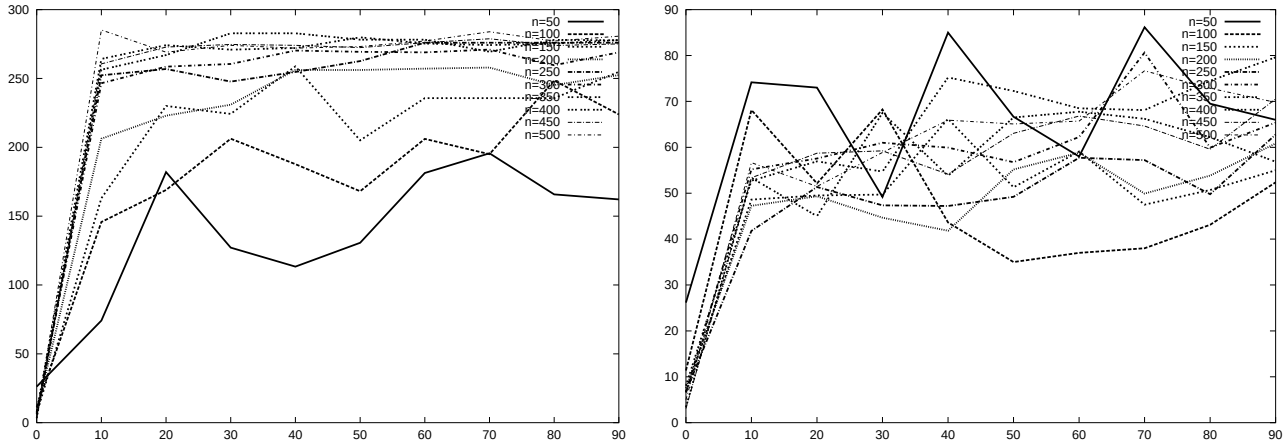

Figure 92: PPS solution deviations for 16-processor hypercube, with and without communication delays

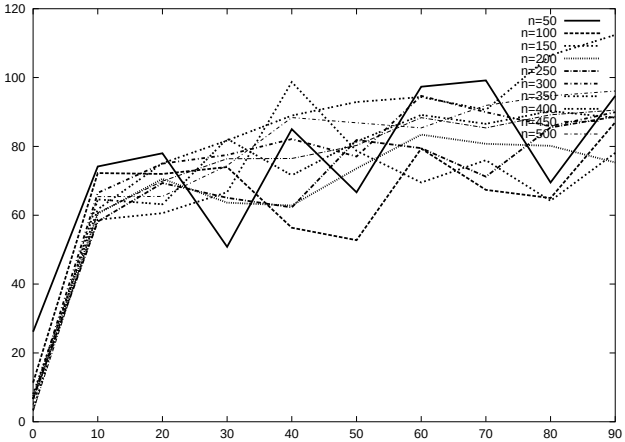

Figure 93: PPS solution deviations for completely connected 16 processors with communication delays

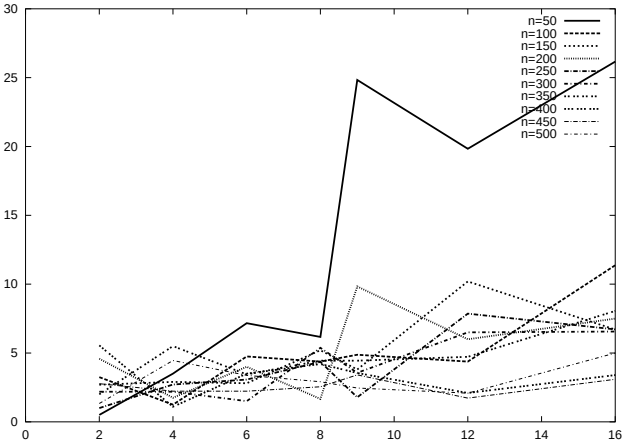

Figure 94: Solution deviations for scheduling independent tasks

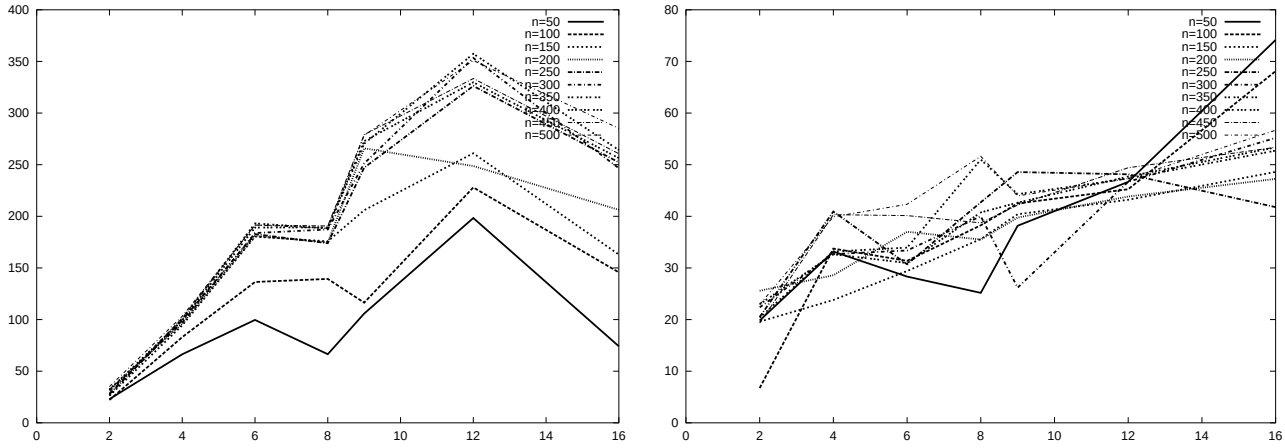

Figure 95: Solution deviations for sparse task graphs $\rho = 10$, with and without communication delays scheduled onto incomplete networks

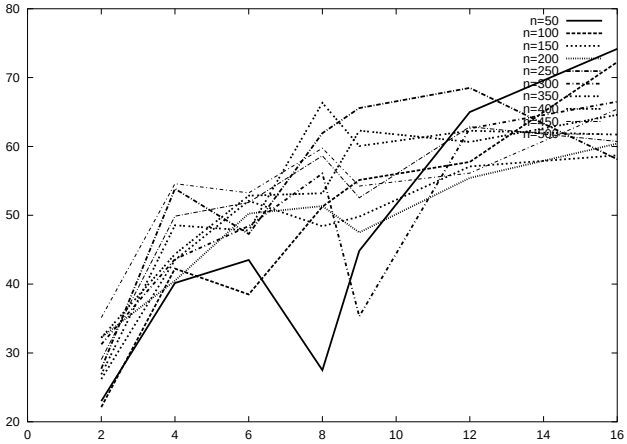

Figure 96: Solution deviations for sparse task graphs $\rho = 10$ with communication delays scheduled onto complete network of processors

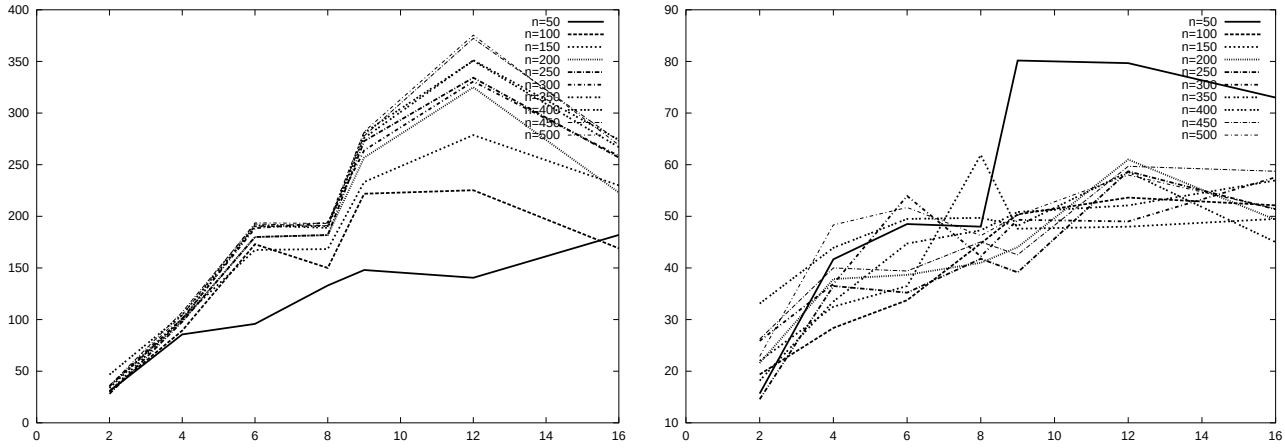

Figure 97: Solution deviations for task graphs with $\rho = 20$, with and without communication delays scheduled onto incomplete networks

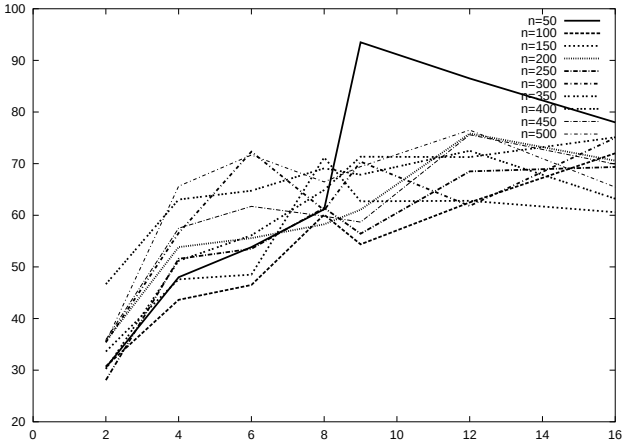

Figure 98: Solution deviations for task graphs with $\rho = 20$ with communication delays scheduled onto complete network of processors

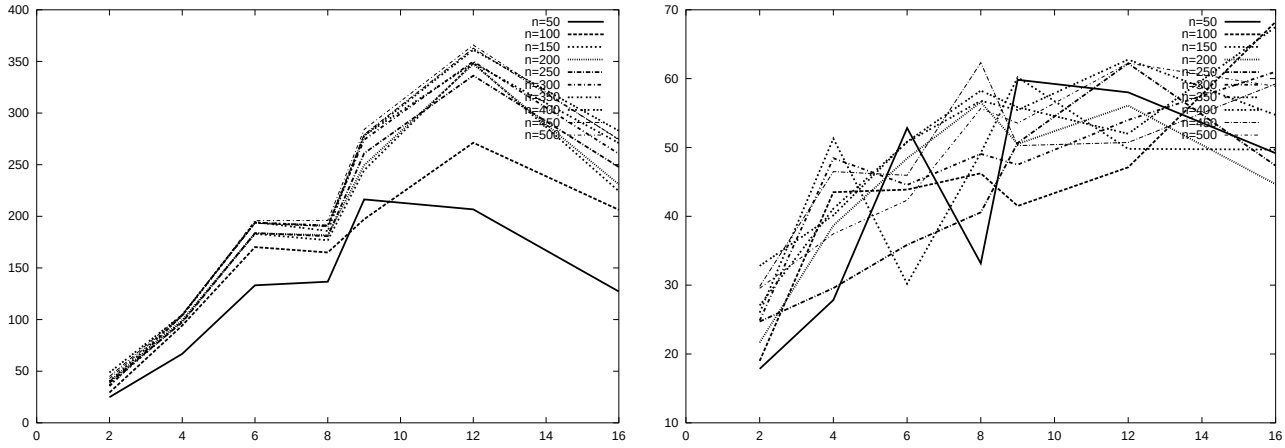

Figure 99: Solution deviations for task graphs with $\rho = 30$, with and without communication delays scheduled onto incomplete networks

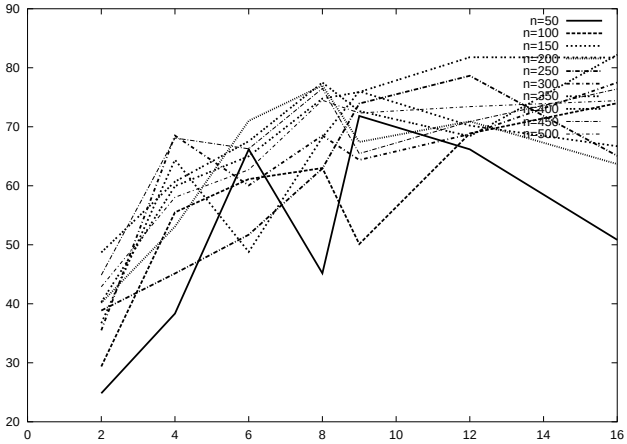

Figure 100: Solution deviations for task graphs with $\rho = 30$ with communication delays scheduled onto complete network of processors

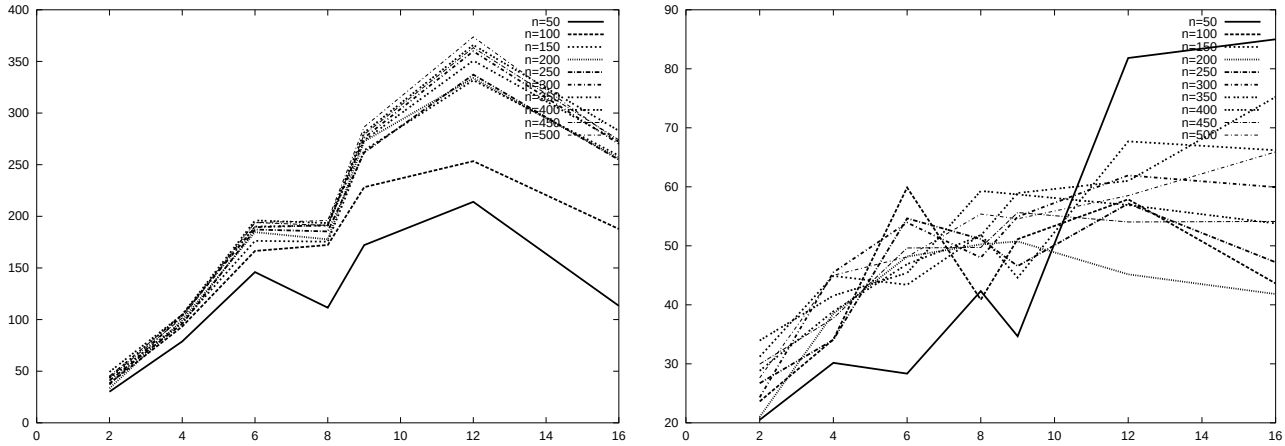

Figure 101: Solution deviations for task graphs with $\rho = 40$, with and without communication delays scheduled onto incomplete networks

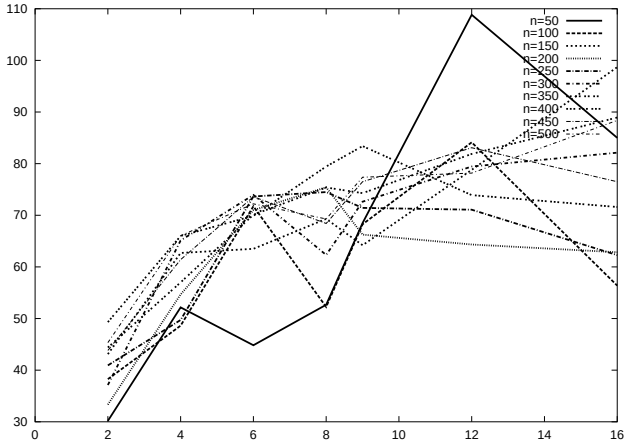

Figure 102: Solution deviations for task graphs with $\rho = 40$ with communication delays scheduled onto complete network of processors

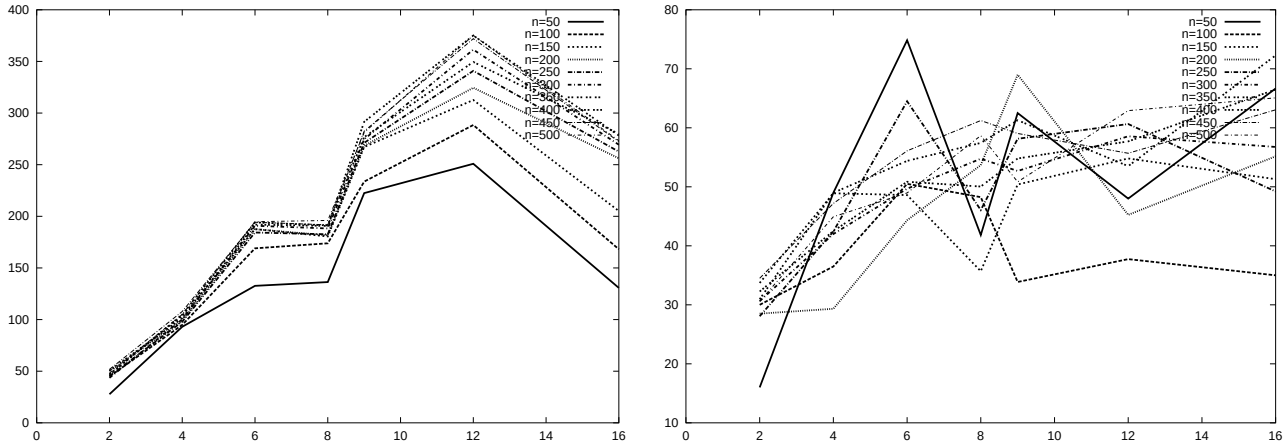

Figure 103: Solution deviations for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

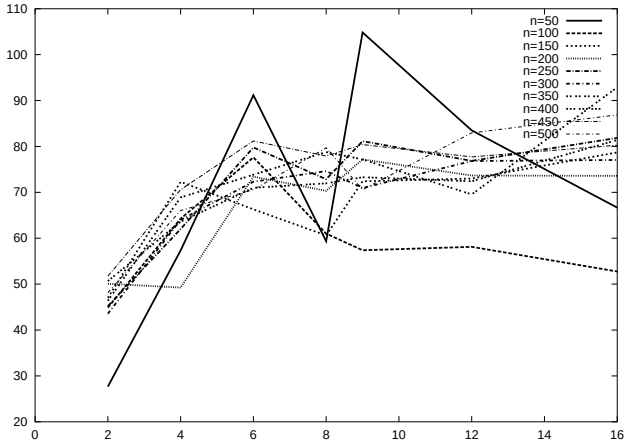

Figure 104: Solution deviations for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

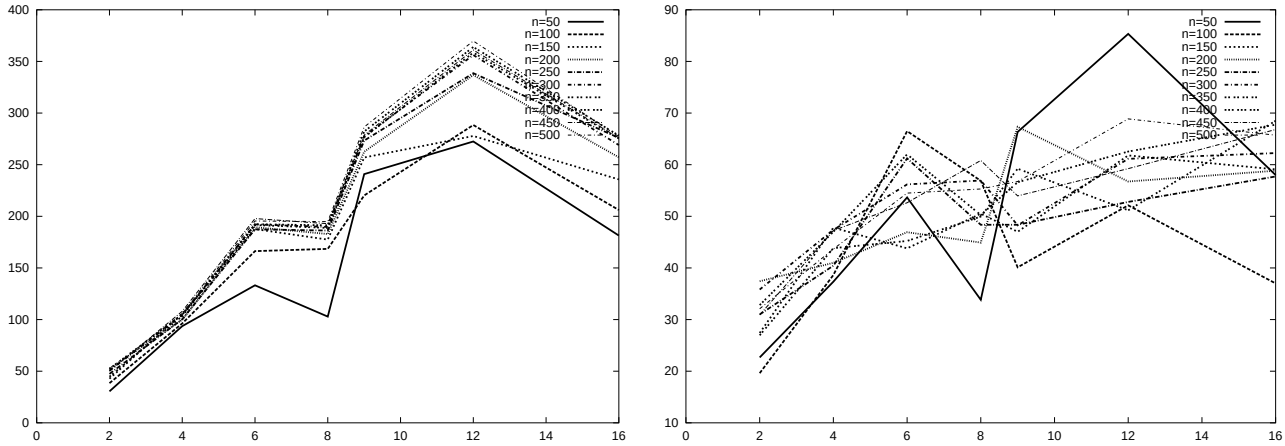

Figure 105: Solution deviations for task graphs with $\rho = 60$, with and without communication delays scheduled onto incomplete networks

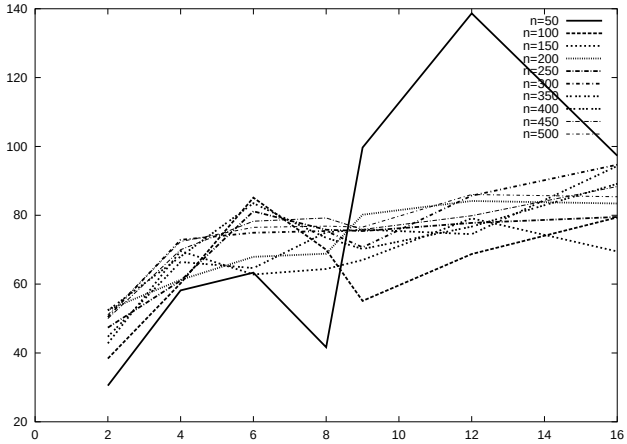

Figure 106: Solution deviations for task graphs with $\rho = 60$ with communication delays scheduled onto complete network of processors

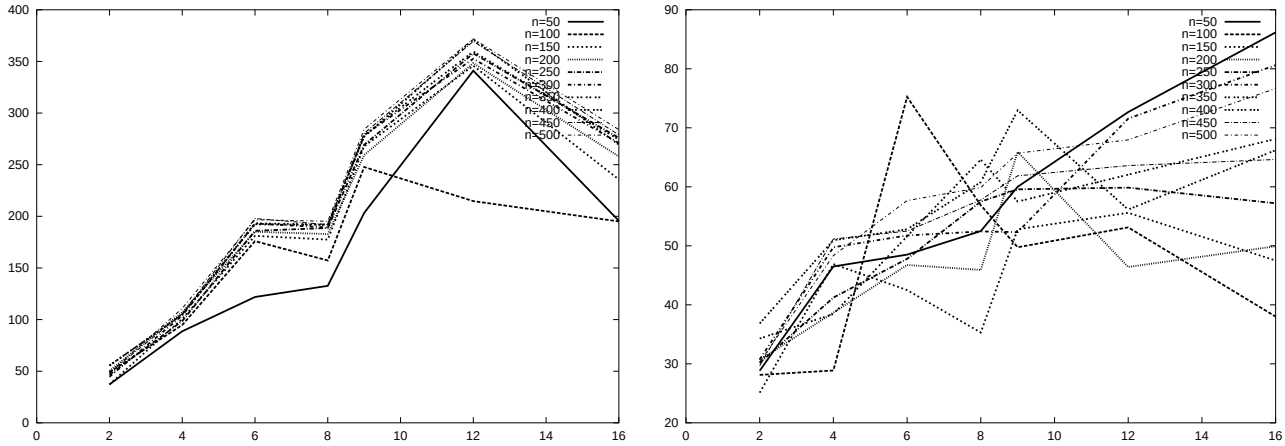

Figure 107: Solution deviations for task graphs with $\rho = 70$, with and without communication delays scheduled onto incomplete networks

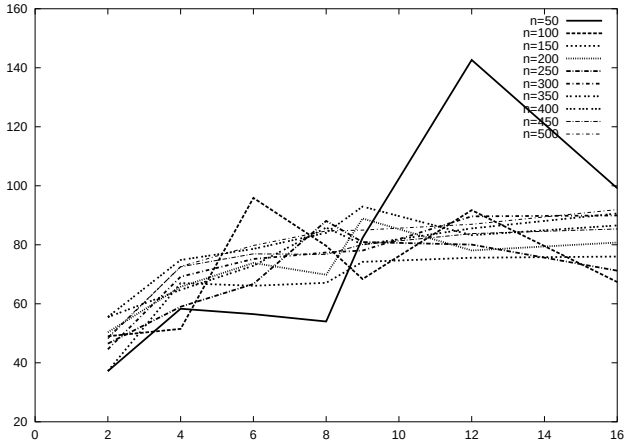

Figure 108: Solution deviations for task graphs with $\rho = 70$ with communication delays scheduled onto complete network of processors

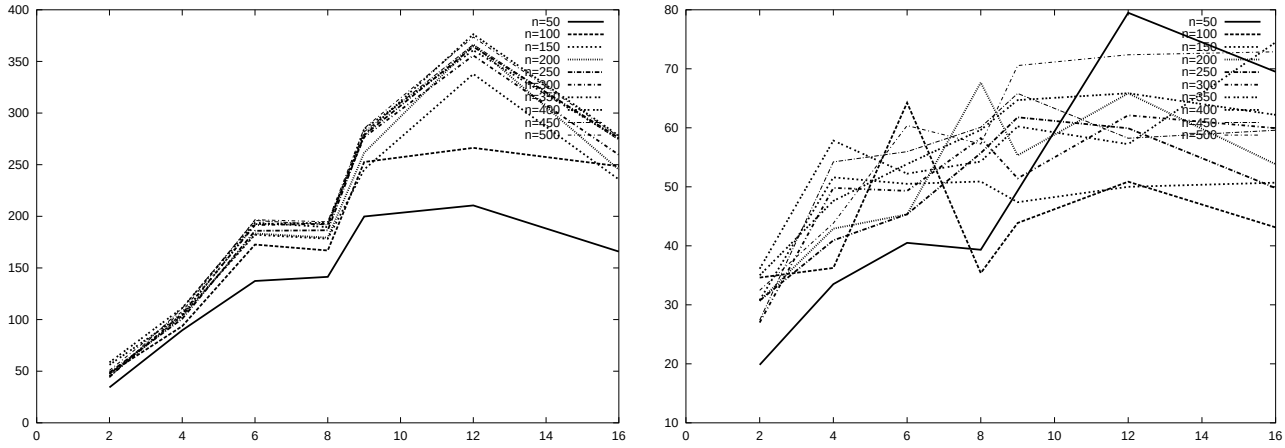

Figure 109: Solution deviations for task graphs with $\rho = 80$, with and without communication delays scheduled onto incomplete networks

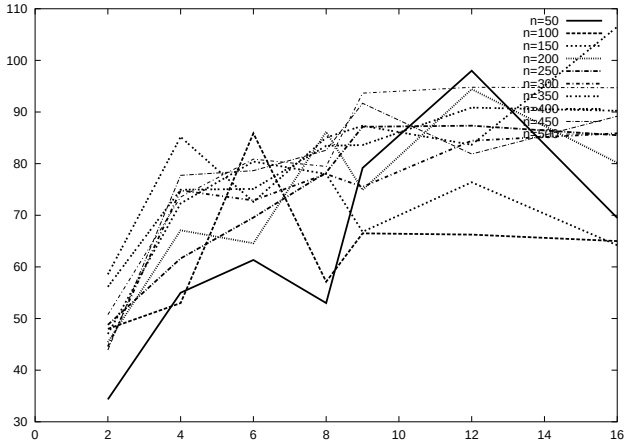

Figure 110: Solution deviations for task graphs with $\rho = 80$ with communication delays scheduled onto complete network of processors

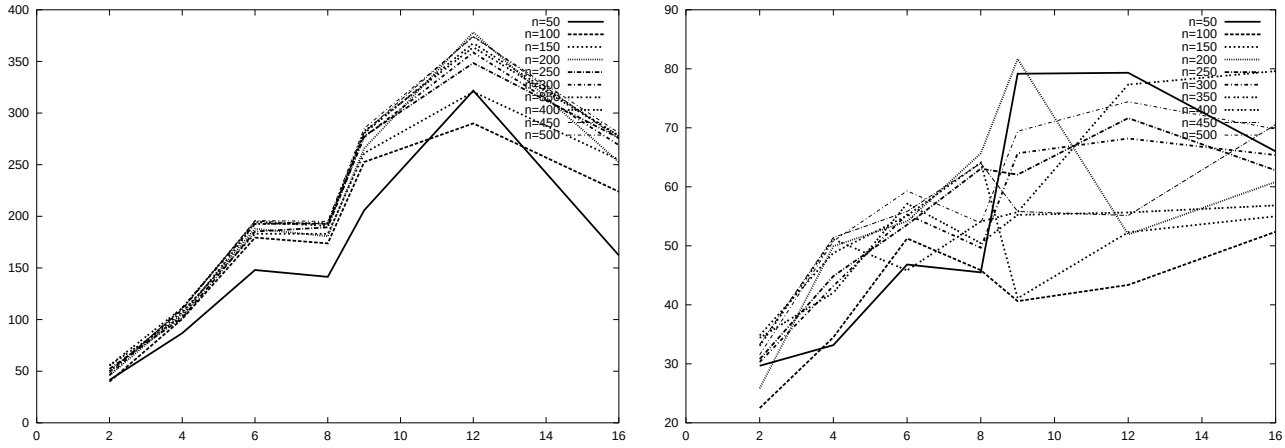

Figure 111: Solution deviations for task graphs with $\rho = 90$, with and without communication delays scheduled onto incomplete networks

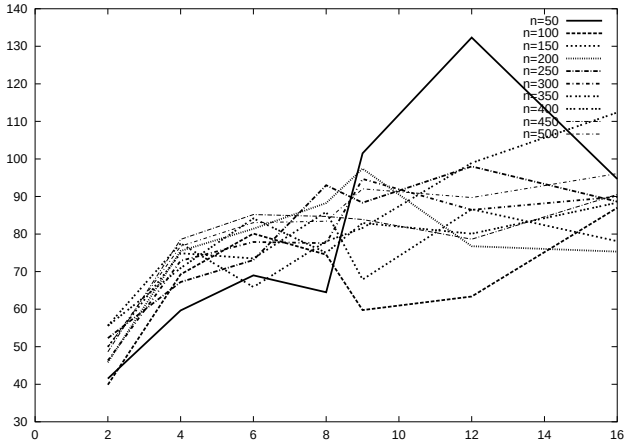

Figure 112: Solution deviations for task graphs with $\rho = 90$ with communication delays scheduled onto complete network of processors

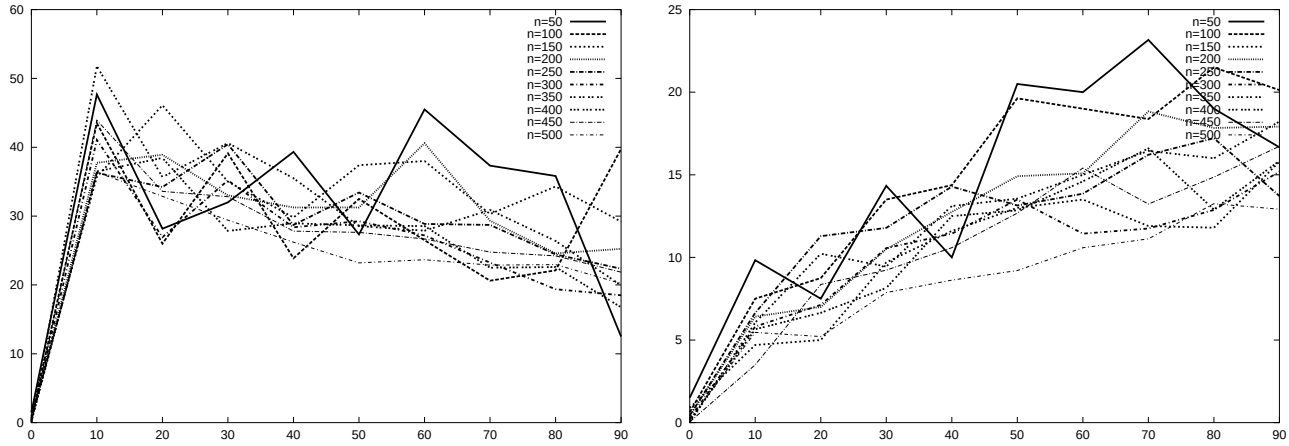

Figure 113: DC solution deviations for 2-processor system, with and without communication delays

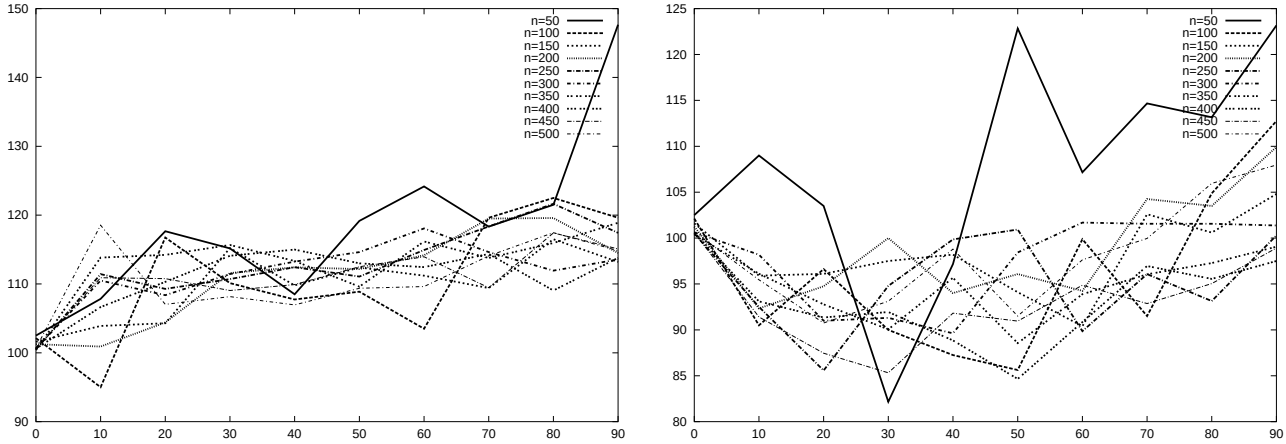

Figure 114: DC solution deviations for 4-processor hypercube, with and without communication delays

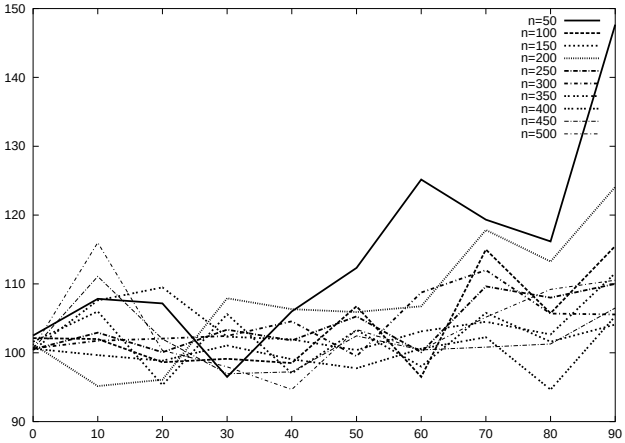

Figure 115: DC solution deviations for completely connected 4 processors with communication delays

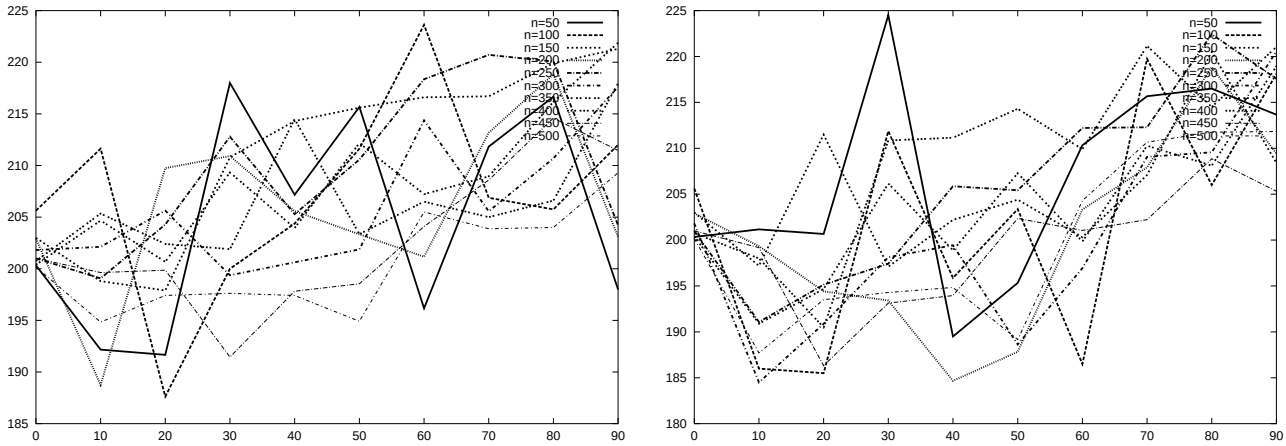

Figure 116: DC solution deviations for 6-processor mesh, with and without communication delays

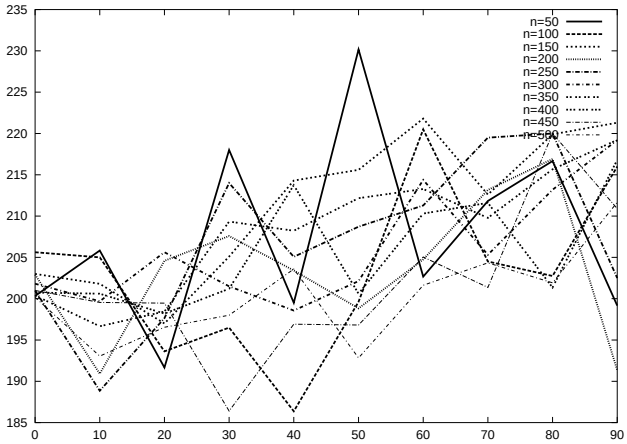

Figure 117: DC solution deviations for completely connected 6 processors with communication delays

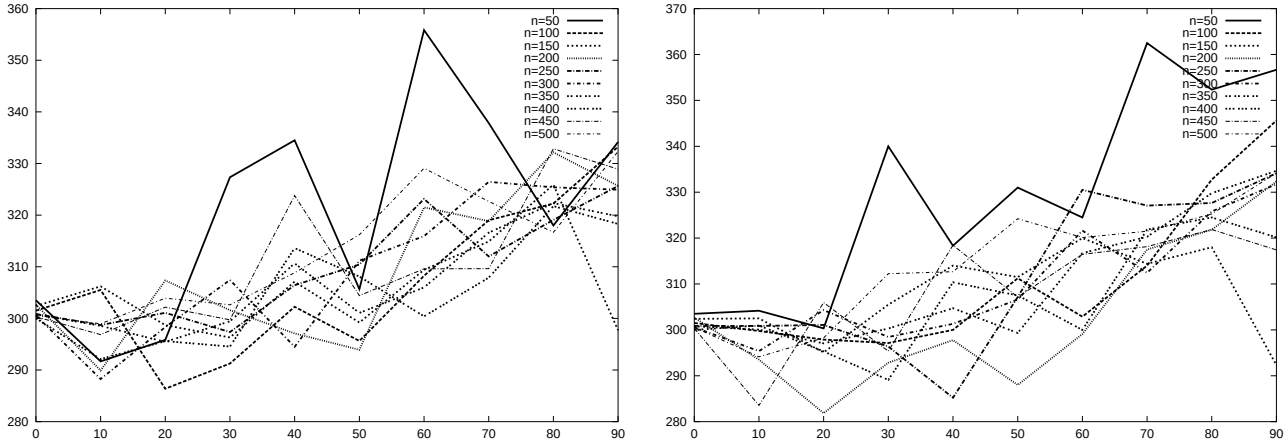

Figure 118: DC solution deviations for 8-processor hypercube, with and without communication delays

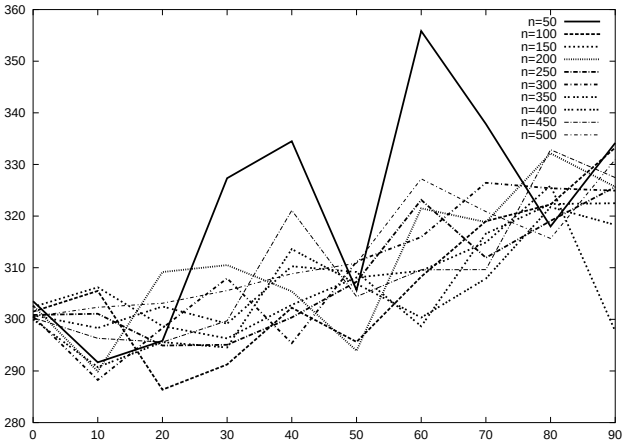

Figure 119: DC solution deviations for completely connected 8 processors with communication delays

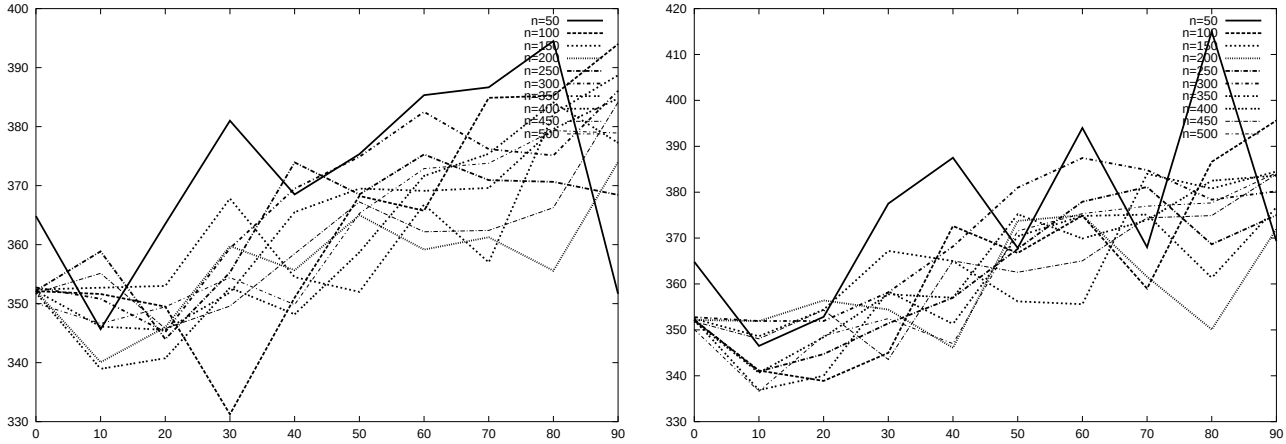

Figure 120: DC solution deviations for 9-processor mesh, with and without communication delays

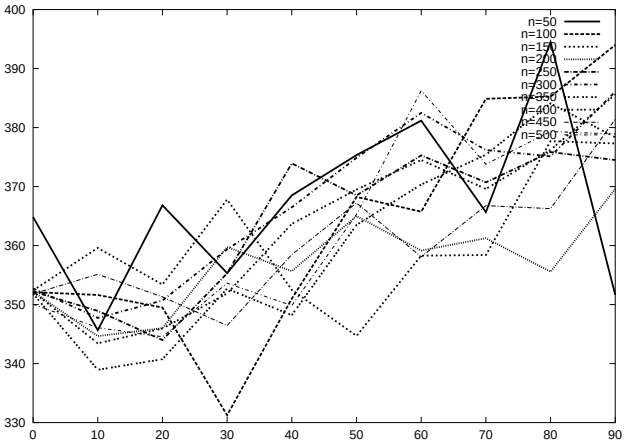

Figure 121: DC solution deviations for completely connected 9 processors with communication delays

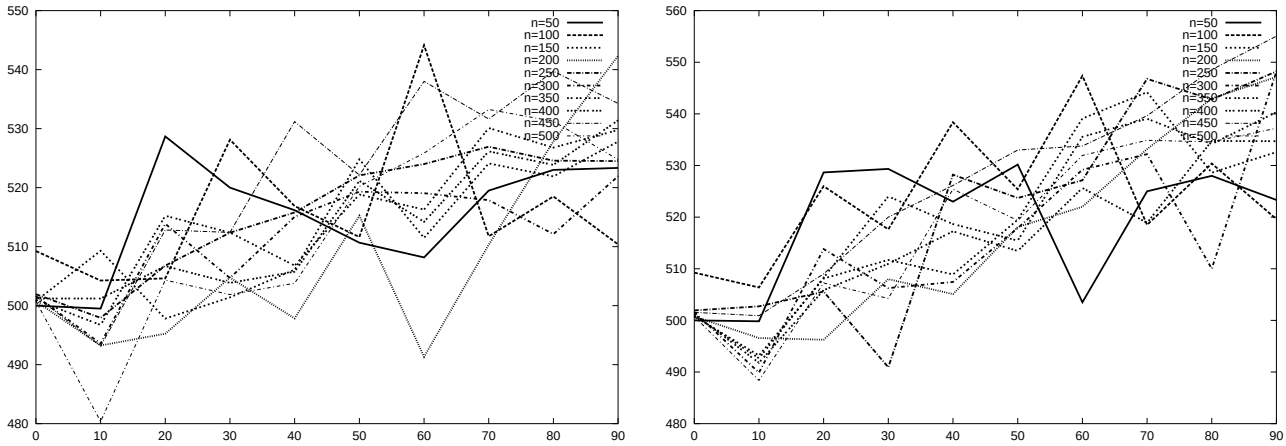

Figure 122: DC solution deviations for 12-processor mesh, with and without communication delays

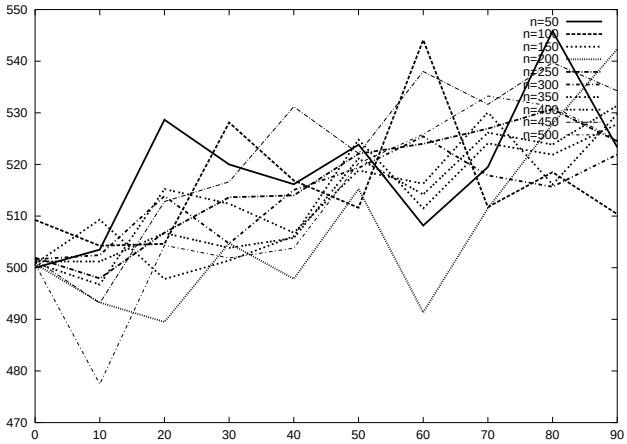

Figure 123: DC solution deviations for completely connected 12 processors with communication delays

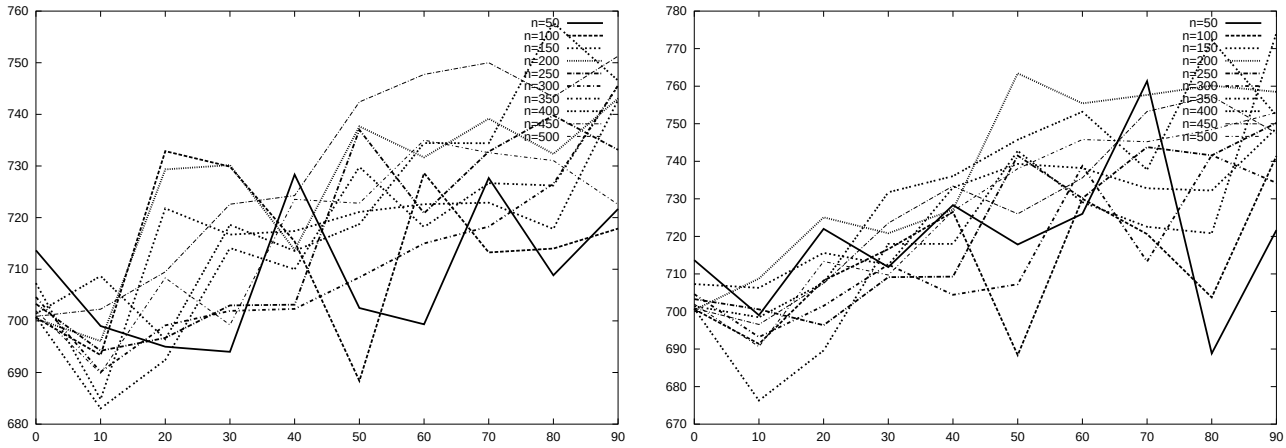

Figure 124: DC solution deviations for 16-processor hypercube, with and without communication delays

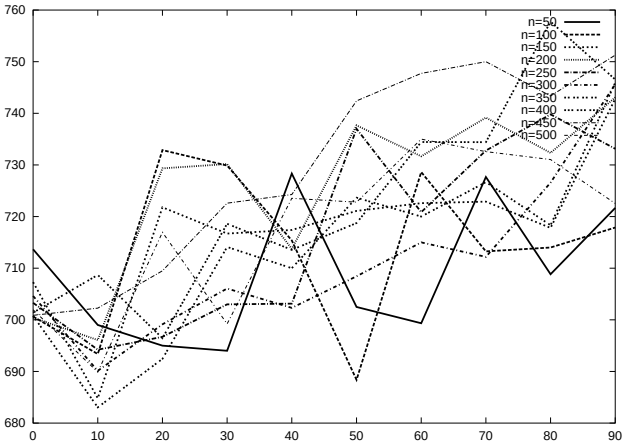

Figure 125: DC solution deviations for completely connected 16 processors with communication delays

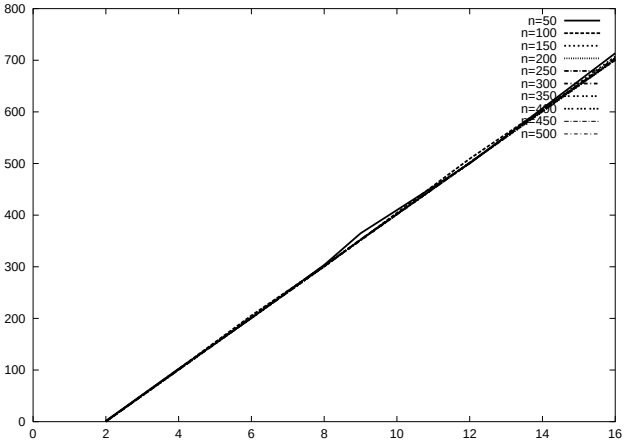

Figure 126: Solution deviations for scheduling independent tasks

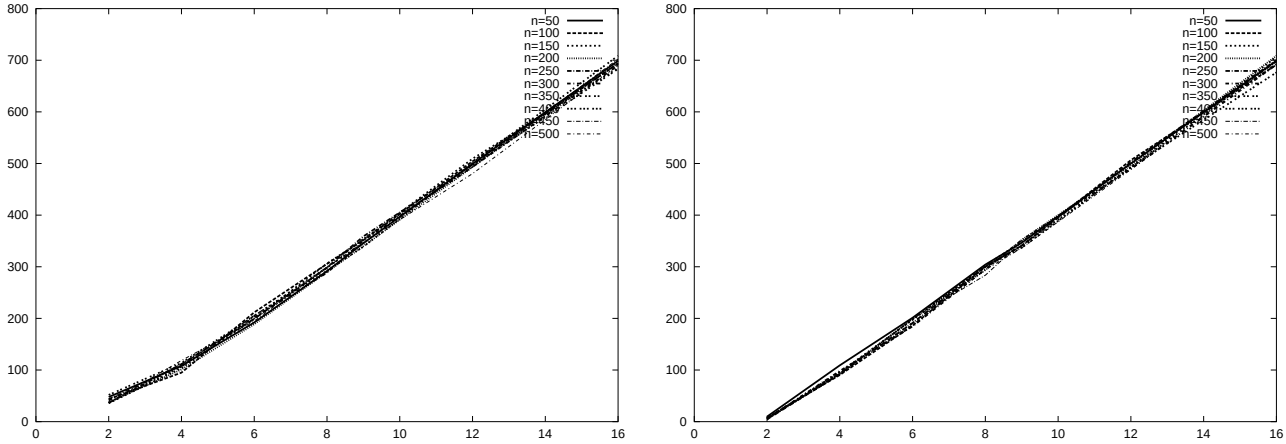

Figure 127: Solution deviations for sparse task graphs $\rho = 10$, with and without communication delays scheduled onto incomplete networks

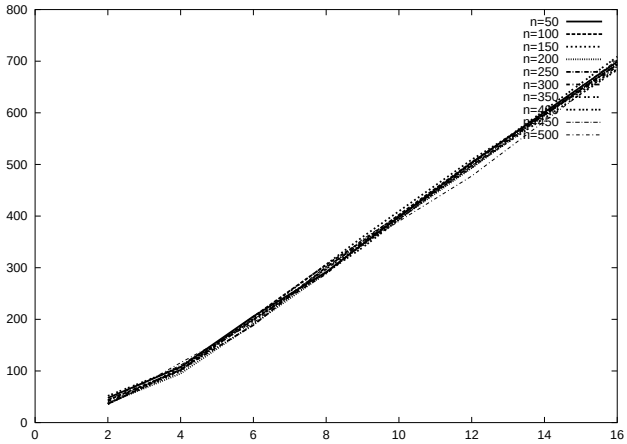

Figure 128: Solution deviations for sparse task graphs $\rho = 10$ with communication delays scheduled onto complete network of processors

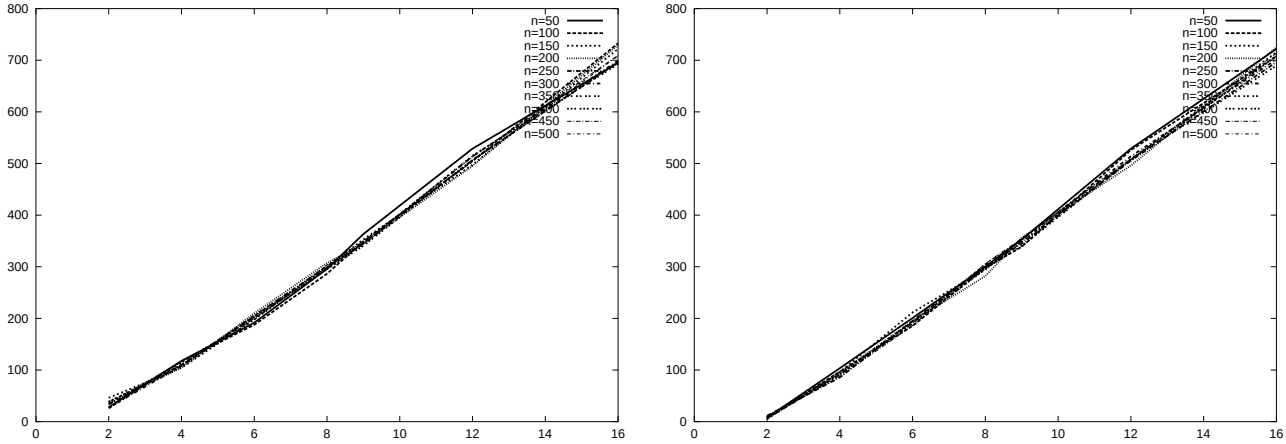

Figure 129: Solution deviations for task graphs with $\rho = 20$, with and without communication delays scheduled onto incomplete networks

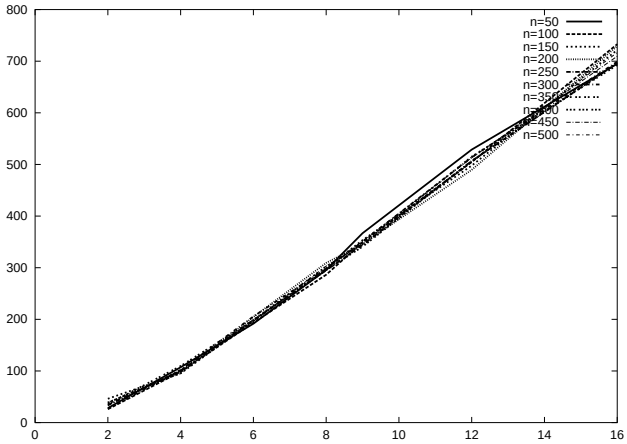

Figure 130: Solution deviations for task graphs with $\rho = 20$ with communication delays scheduled onto complete network of processors

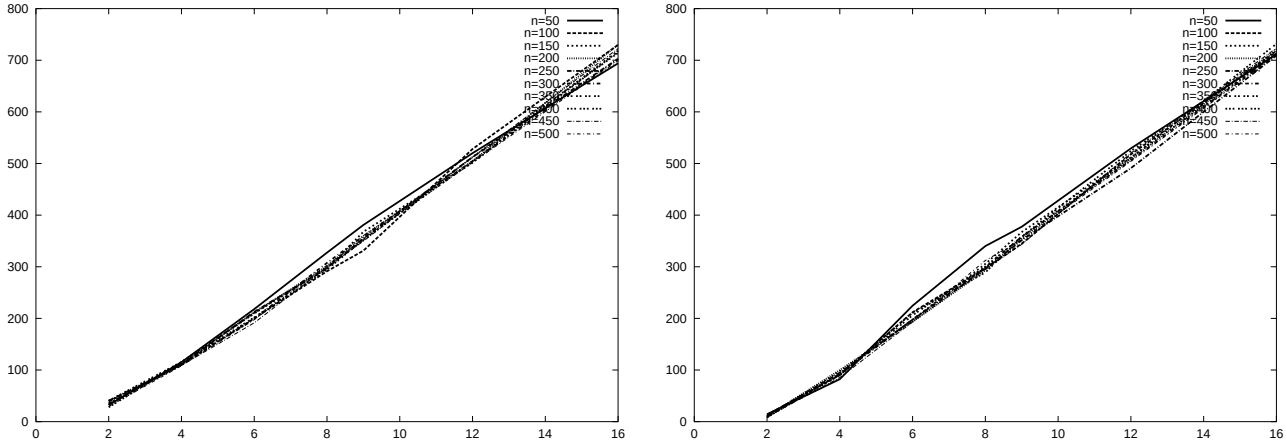

Figure 131: Solution deviations for task graphs with $\rho = 30$, with and without communication delays scheduled onto incomplete networks

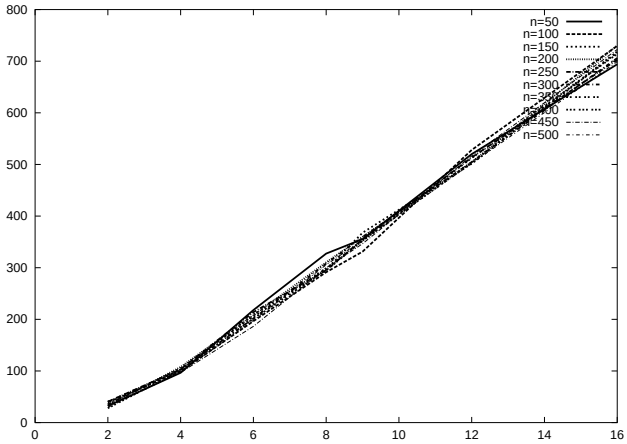

Figure 132: Solution deviations for task graphs with $\rho = 30$ with communication delays scheduled onto complete network of processors

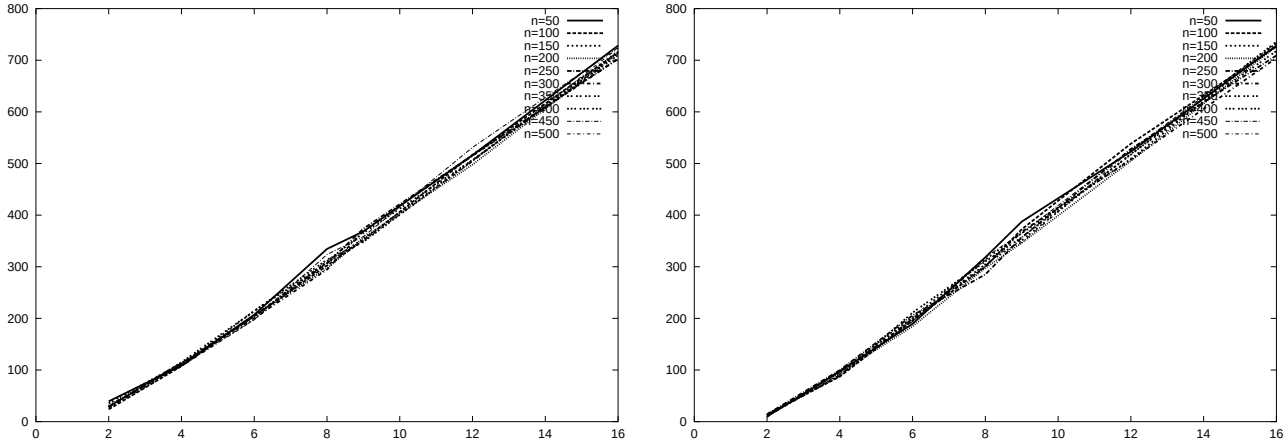

Figure 133: Solution deviations for task graphs with $\rho = 40$, with and without communication delays scheduled onto incomplete networks

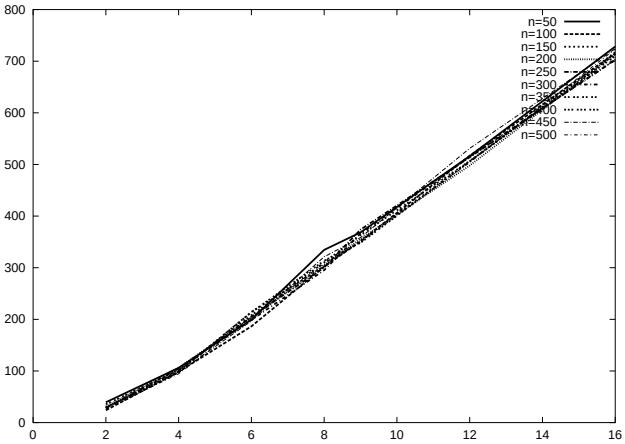

Figure 134: Solution deviations for task graphs with $\rho = 40$ with communication delays scheduled onto complete network of processors

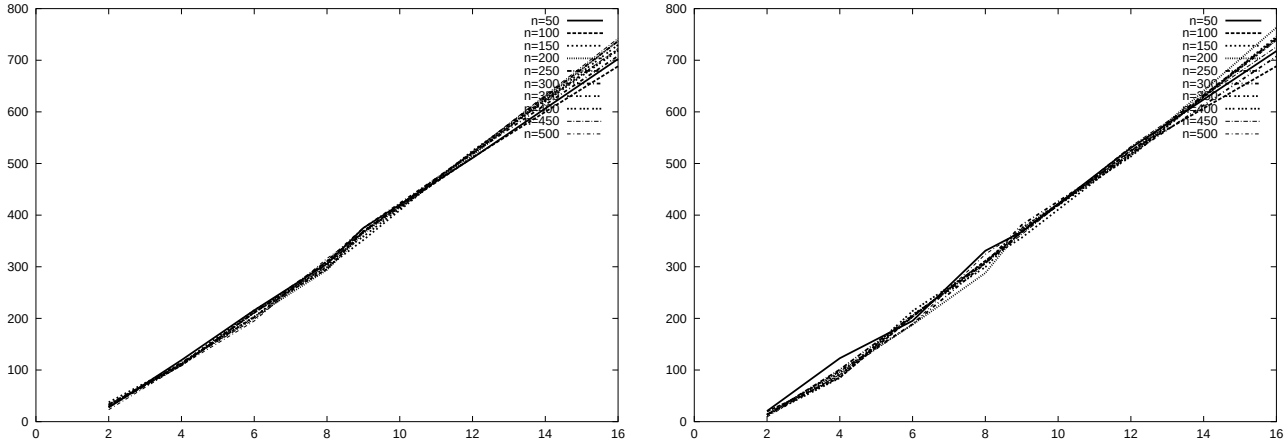

Figure 135: Solution deviations for task graphs with $\rho = 50$, with and without communication delays scheduled onto incomplete networks

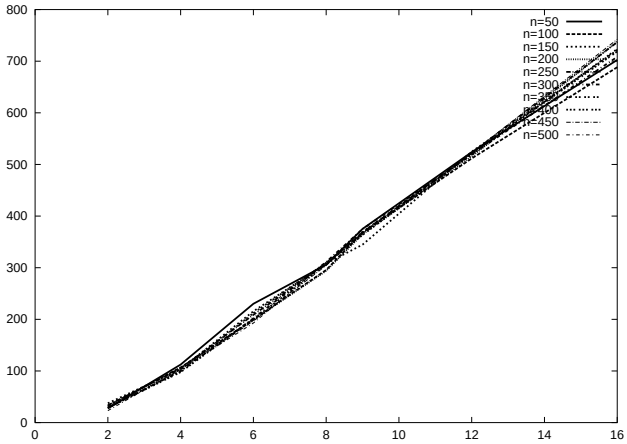

Figure 136: Solution deviations for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

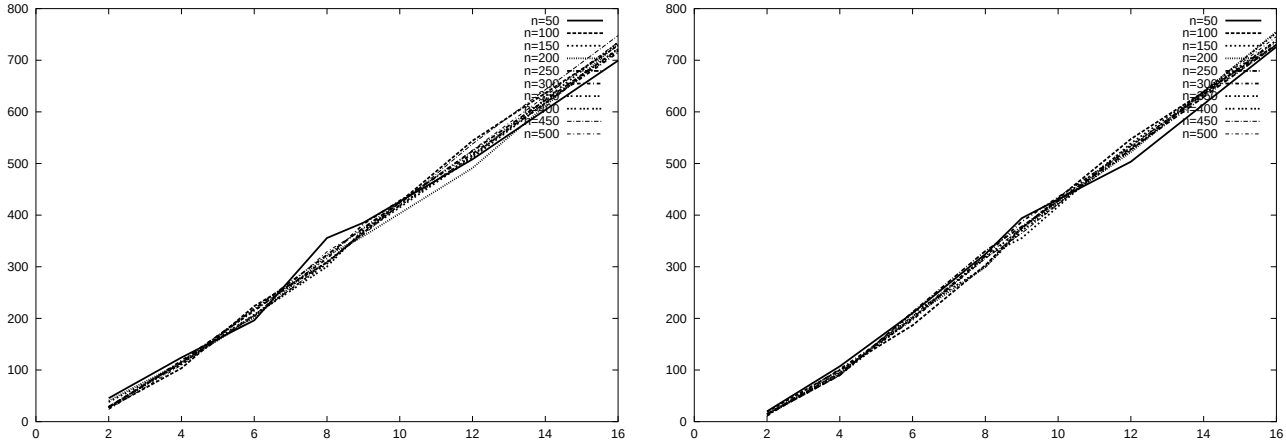

Figure 137: Solution deviations for task graphs with $\rho = 60$, with and without communication delays scheduled onto incomplete networks

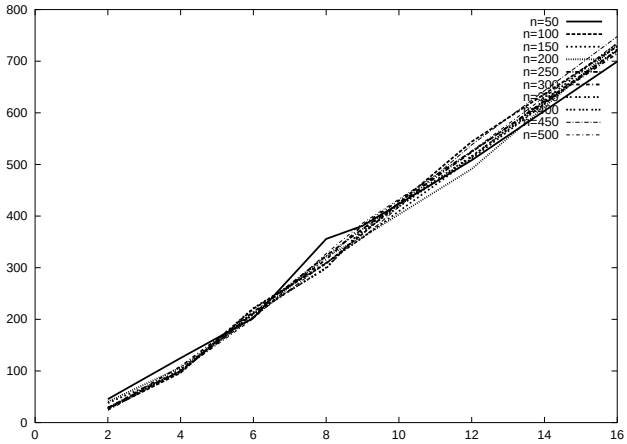

Figure 138: Solution deviations for task graphs with $\rho = 60$ with communication delays scheduled onto complete network of processors

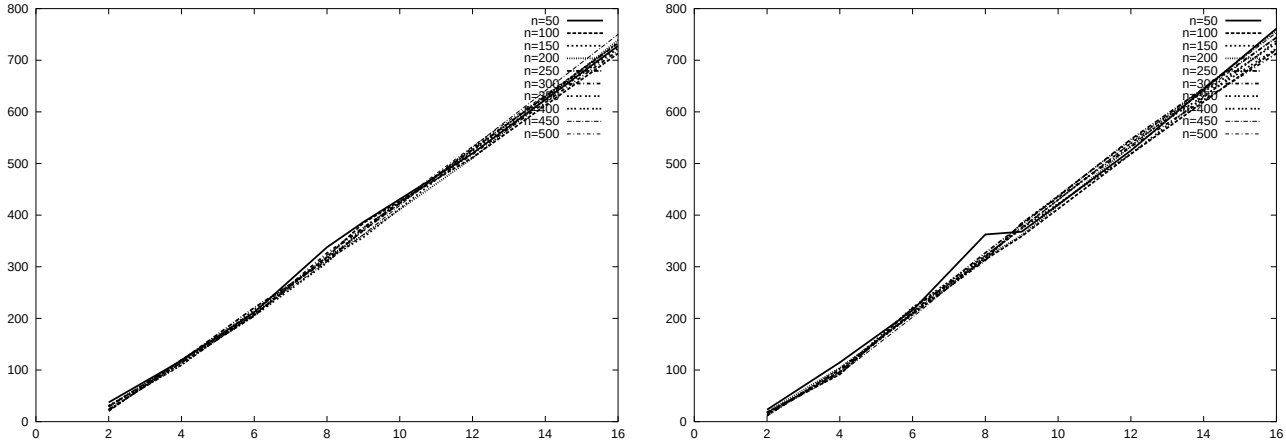

Figure 139: Solution deviations for task graphs with $\rho = 70$, with and without communication delays scheduled onto incomplete networks

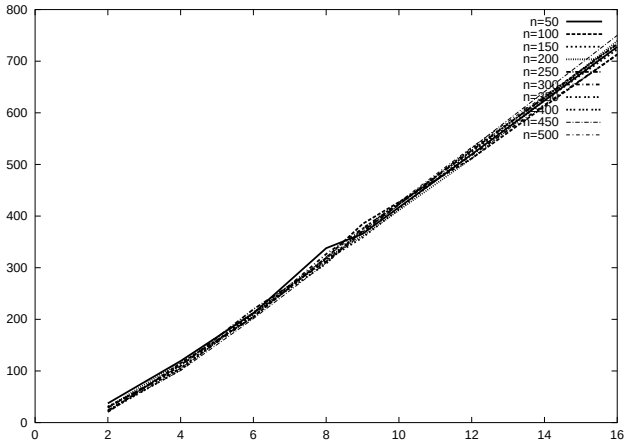

Figure 140: Solution deviations for task graphs with $\rho = 70$ with communication delays scheduled onto complete network of processors

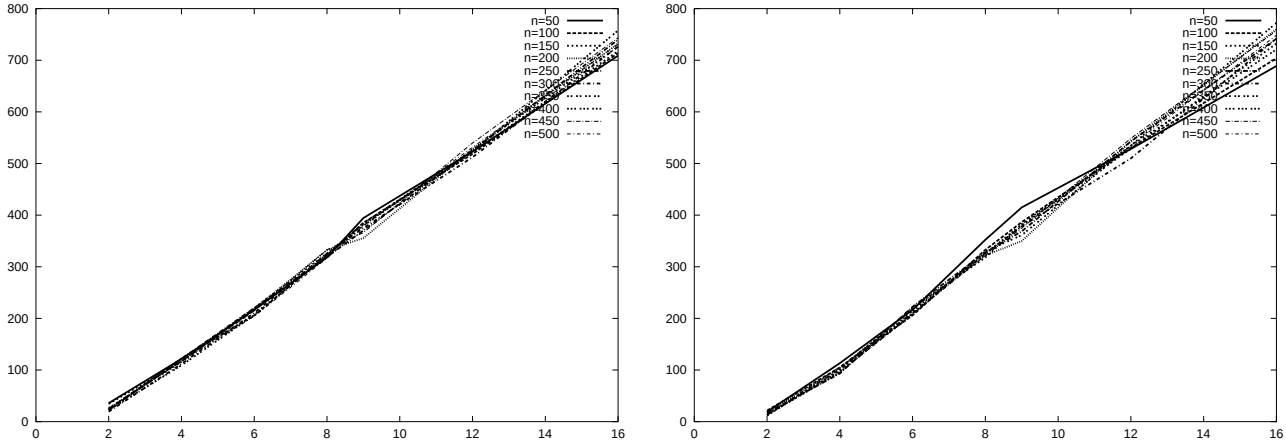

Figure 141: Solution deviations for task graphs with $\rho = 80$, with and without communication delays scheduled onto incomplete networks

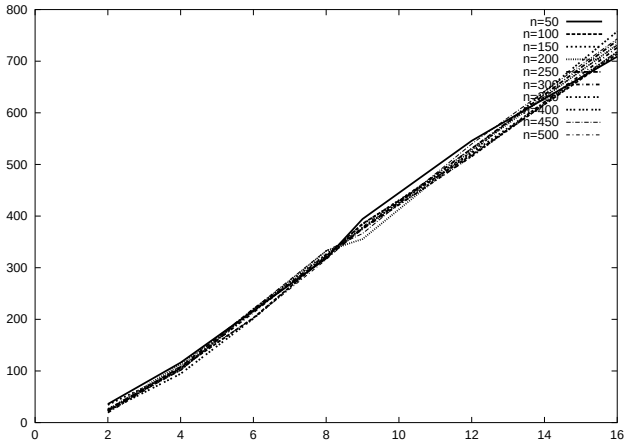

Figure 142: Solution deviations for task graphs with $\rho = 80$ with communication delays scheduled onto complete network of processors

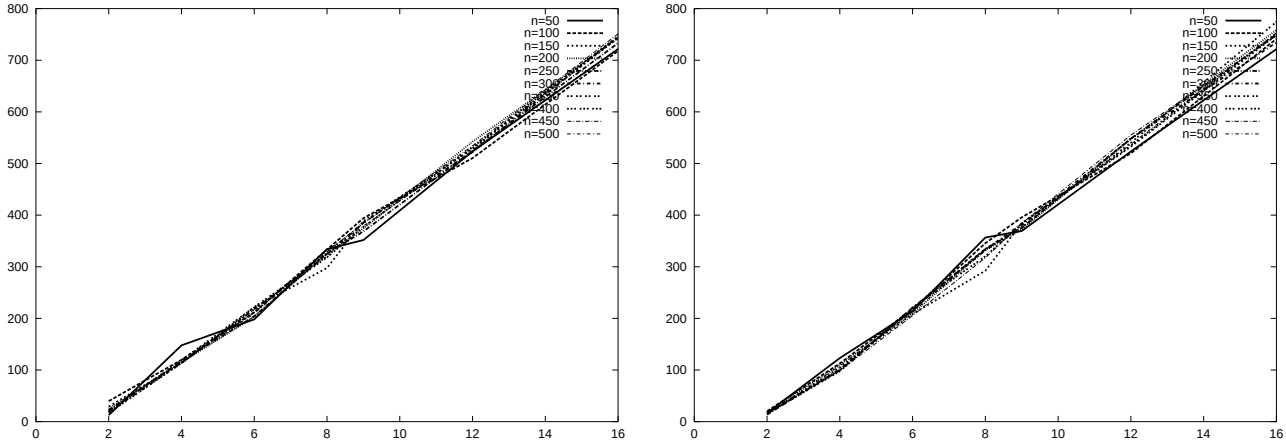

Figure 143: Solution deviations for task graphs with $\rho = 90$, with and without communication delays scheduled onto incomplete networks

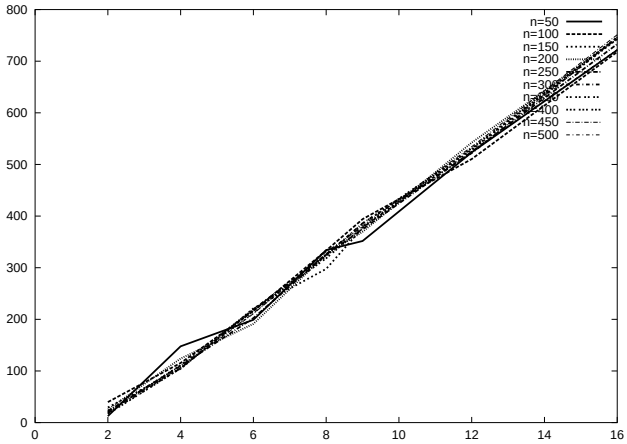

Figure 144: Solution deviations for task graphs with $\rho = 90$ with communication delays scheduled onto complete network of processors