Bibliography research

Finding an ILP formulation for our problem was surprisingly difficult. Most of the bibliography we found deal with more complex versions of the problem, such as:

- Different types of processors [1] [2]
- Communication delays between the tasks [3]
- Independent tasks (with no precedence constraints) [4]

Unfortunately, it was not possible for us to simplify one of those formulations by just "relaxing" their constraints to adapt it for our problem.

[5] proposes a solution to the scheduling problem that requires that the binding process (assignment of each task to a particular processor) be performed prior to solving the ILP formulation. In other words, it is not a complete ILP formulation for our problem, since it does not look for a solution in the complete problem domain, but only in a particular predefined binding. In addition, this formulation does not solve the optimization version, but only the decision version of the problem (whether the finish time of the last task is smaller than a given deadline). They remark that it was surprisingly non-trivial to represent this problem as a Binary Linear Program.

[6] proposes a formulation for the optimization problem, and it does not require a predefined binding. It differs from our problem in that it defines communication delays (of unit length) between the tasks, and that the number of processors is unbounded.

[7] expands on the results of [6] by adding a constraint to bound the number of processors.

ILP Formulation

First Formulation

For this section we have developed two different formulations each fulfilling a different purpose. One formulation is based on [6]. In it we have removed communication delays, and introduced a constraint on the number of processors. The purpose of the below formulation is establishing a lower bound on execution in the unbound processor case. We did not use the LP version of the below formulation as the bound it would produce would be meaningless. The second formulation is used to evaluate performance in the bounded number of processor case. A description of the first formulation is given below.

Constants:

- N: number of tasks
- Q: number of processors
- M: large positive integer

Variables:

- T: makespan (integer)
- t_i: start time of task i (N integer variables)
- p_i: processor where task i is processed (N integer variables)

- Y_{ij}: activation of the i,j either-or constraint (N^2 binary variables)

Objective: minimize T

Subject to:

- For all i in N: $t_i + 1 <= T$ (constraint for defining the makespan)
- For all i in N: $t_i >= 0$
- For all edges i->j: $t_i + 1 <= t_j$ (constraint on partial order, but with no communication delay)
- For all i in N: $p_i <= Q - 1$ (constraint on number of processors)
- For all i in N: $p_i >= 0$
- For i in 2…N, j in 1…i-1: $p_i + t_i*Q \neq p_j + t_j*Q$ (no two tasks assigned at the same time to the same processor). This can be rewritten as the following either-or constraints:
  $M*Y_{ij} + p_i + t_i*Q >= p_j + t_j*Q + 1$
  $p_i + t_i*Q =< p_j + t_j*Q - 1 + M*(1 - Y_{ij})$

The last constraint can be understood as having a "fingerprint" for each task, based on when and where they are scheduled. Each processor can process only one task at a time, so it is required that all tasks have a different fingerprint from each other, which can be specified with such linear inequality constraint.

Lower bound:

We believe removing the integrality constraint to obtain the LP bound would not be very useful for this problem. Conceptually, such relaxed problem would represent a model that is very different from the real world, where all the variables need to have discrete values, which would probably produce a bound that is very far from being tight.

Instead of working with the LP formulation, we decided to look for a tight bound by relaxing the constraint on the number of processors. We first replaced such constraint with the following:

- For all i in N: $p_i <= W - 1$

Where W is the width of the partial order, and is defined as the size of the maximum antichain, which can be summarized as the size of the maximum set of tasks that are concurrent and could be processed at the same time. However, this new relaxed constraint did not provide practical benefits for computing the lower bound, since the solver could not solve the instances quickly.

Therefore, for the relaxed formulation we removed this constraint, and solved the problem with unbounded number of processors. This way, the lower bound could be solved very quickly for all the instances.

Second Formulation

The second formulation is from [7] in the bounded processor case where the number of processors is less than the maximum width of the instance. Width is defined as the maximum number of tasks executable concurrently in the graph. This second formulation is what is used when assessing benchmark solution performance and number of instances solved. An automated task graph formulation generator was built using Pyomo. The generator can take arbitrary task graphs and

generates an ILP model that conforms to the below formulation. Pyomo is solver independent, but all results generated below were generated using Gurobi. Below is a breakdown of the formulation from [7].

$$
\begin{array}{lll}
(1) & x_j \geq x_i + 2 - w_{ij}, & i,j \in N, i \vdash j \\
(2) & x_{n+1} \geq x_i + 1, & i \in M_{\prec}^{-} \\
 & & \\
(5) & \sum_{i \in N} w_{i,n+1} \leq q, & \\
(6) & w_{ij} \in \{0,1\}, & i,j \in N, i \prec j \\
(7) & x_i \geq 0, & i \in M_{\prec}^{+} \\
 & & \\
(8) & x_j \geq x_i - \alpha_{ij}\overline{w}_{ij} + 1, & i,j \in N, i \parallel j \\
(9) & w_{0j} + \sum_{i \prec j} w_{ij} + \sum_{i \parallel j} w_{ij} = 1, & j \in N \\
(10) & w_{i,n+1} + \sum_{i \prec j} w_{ij} + \sum_{i \parallel j} w_{ij} = 1, & i \in N
\end{array}
$$

Variables

Each task is given a start time variable xi. Additionally, a final artificial task is added to the task graph with start time x_n+1. It's value indicates the makespan. All tasks that have no successors must have a start time that precedes this final artificial node by at least 1 unit time. Each pair of tasks share an ordering variable w_ij that indicates whether both tasks are executed on the same processor in the order i->j with no tasks in between. W_ij is not limited to tasks that share a precedence relationship. In fact, concurrent tasks that have no precedence relationship at all can be executed sequentially on the same processor in the order w_ij.

The objective for the above formulation is to minimize x_n+1.

Constraint 1 guarantees task precedence by applying a transitive reduction to the task graph and then ensuring that tasks that have a precedence relationship never execute in an order that violates that relationship. This constraint also includes a communication overhead in addition task execution time. This overhead has been removed from our implementation of the formulation.

Constraint 2 guarantees that the makespan (start time of the last artificially added node) exceeds the last set of tasks in the task graph that have no successors.

Constraint 5 guarantees that the number of tasks that can execute concurrently before the last artificial end node of the graph cannot exceed the number of processors available. This constraint when combined with constraints 8, 9, and 10 guarantees the total amount of concurrently executing tasks does not exceed q which is the number of available processors. Generally, constraints 9 and 10 impose on ordering of tasks on available processors. Constraint 5 limits the maximum ordering that can occur for each processor to q. This constraint is effectively "propagated" by constraints 9 and 10 throughout the task graph.

Constraint 6 guarantees that task ordering w_ij is a binary variable (either the tasks are ordered i->j or not, a real value is meaningless here).

Constraint 7 guarantees that all tasks that have no predecessors must execute after time 0

Constraint 9 guarantees that each task can have only one predecessor if that predecessor is executed on the same processor. The predecessor can be a task that is has a precedence relationship with the task in that it proceeds it in the task graph, or it can be a task that can be executed concurrently with no precedence related implications. Additionally, that predecessors can be an artificial start task that is not in the original graph.

Constraint 10 guarantees that each task can only have one successor if that successor is executed on the same processor. The successor can be a task that has a precedence relationship with the task in that it succeeds it in the task graph, or it can be a task that can be executed concurrently with no precedence related implications. Additionally, that successor can be the artificial end task that is not in the original graph.

Constraint 8 guarantees that if two tasks are ordered one after another on the same processor then they must execute sequentially and not concurrently. If no ordering exists between those tasks then this constraint is deactivated by the term alpha_ij rendering the constraint inactive.

Results

Benchmark performance

Fig. 1 illustrates the relationship between the number of instances solved and the properties of each instance in terms of node count and number of available processors. All instances were generated with a constant edge probability = 0.5 using the layer-by-layer technique described in previous iterations of this report. Surprisingly, the performance of the ILP model improves with the availability of more processors. This is understandable given that increased opportunities for concurrency limit the search space considerably by easing constraint 5. Generally instances were optimal results are always found given enough time are less than 30 nodes .

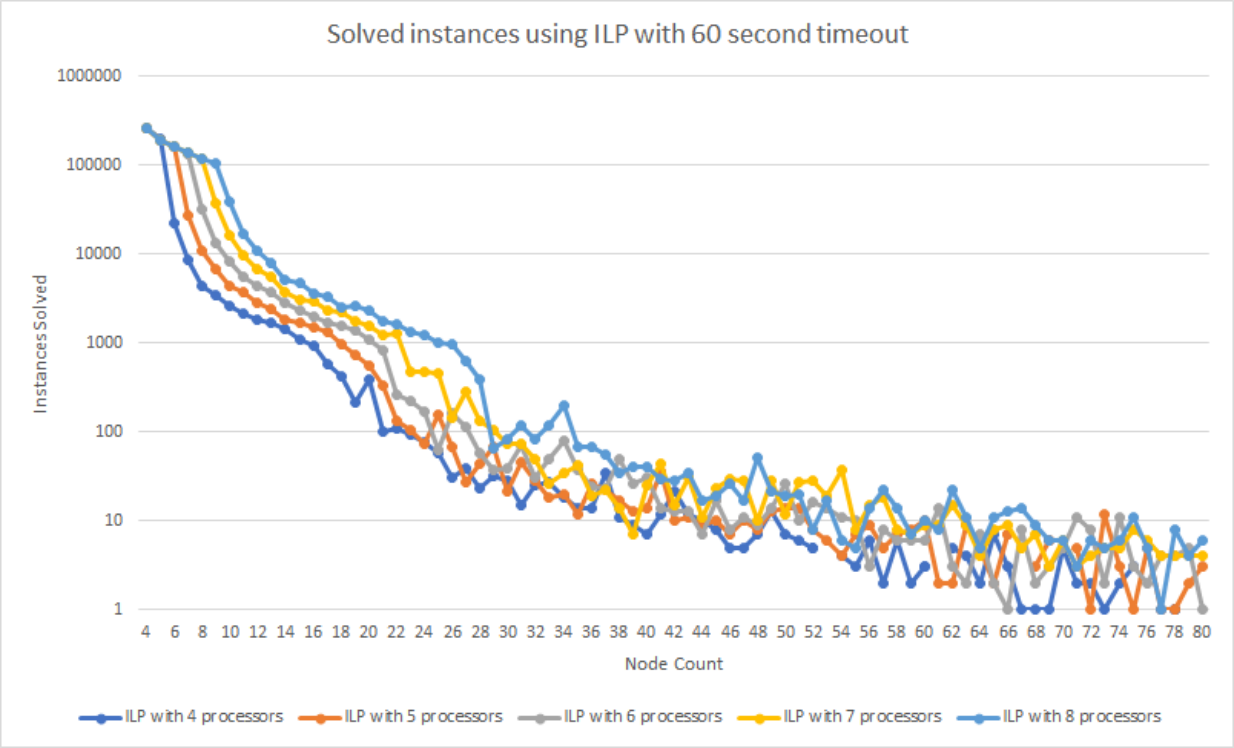

Fig. 1 Instances solved under a 60 second timeout using the second ILP formulation

Quality of results

The below figures 2 through 7 illustrate the second formulation's performance in comparison to the unbounded processors lower bound, exhaustive search, and the greedy approach described in previous iterations of this report. A sweep of instances from 5 nodes to 29 nodes was used to evaluate the performance of the ILP model. The probability of generating edges in the graphs was set to a constant 0.20. The behavior of the ILP is sensitive to the sparsity of the task graph, an exploration of this relationship is available in the next section. Additionally, if an instance is too difficult for the solver and the timeout is sufficient, the solver can produce a solution that is not guaranteed to be optimal. In practice this solution is always superior to the greedy and exhaustive approach.

For instances where exact solutions are available either via the ILP model or via exhaustive search the greedy algorithm performed well. Performance of greedy relative to other techniques was intimately tied to the number of available processors. Effectively the greedy algorithm acts as an upper bound on performance when compared to other techniques. Exhaustive search can be an upper bound even when it times out and produces a sub-optimal result, but comparatively, the greedy algorithm is a tighter upper bound on performance. As can be seen in figures 2 through 7, the gap between the greedy algorithm and lower bound on performance decreases with more processors. Similarly, the lower bound gets tighter with more processors which is understandable because the lower bound assumed an unbounded number of processor so any increase in processors causes the bound to tighten. Additionally, the likelihood that the solver produces an exact solution goes up with more processors.

The ILP model vastly outperforms exhaustive search in regardless of whether it finds an optimal or a suboptimal solution. Additionally, based on Fig. 1 it can solve many more instances at much larger sizes

than exhaustive search. This is likely due to the method in which it explores the scheduling search space. The ILP model schedules start times directly were as the exhaustive algorithm relies on the binding-scheduling split mentioned in previous iterations of this report.

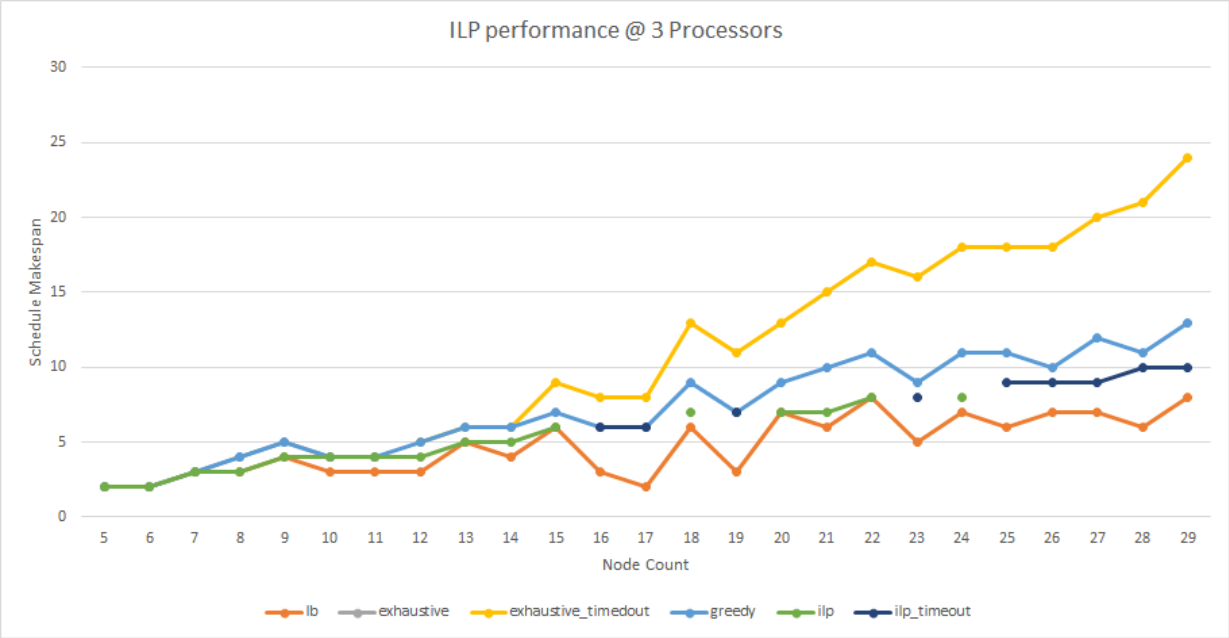

Fig 2. ILP performance with 3 processors against previous techniques @ timeout = 10 seconds with average greedy-lb gap = 2.56

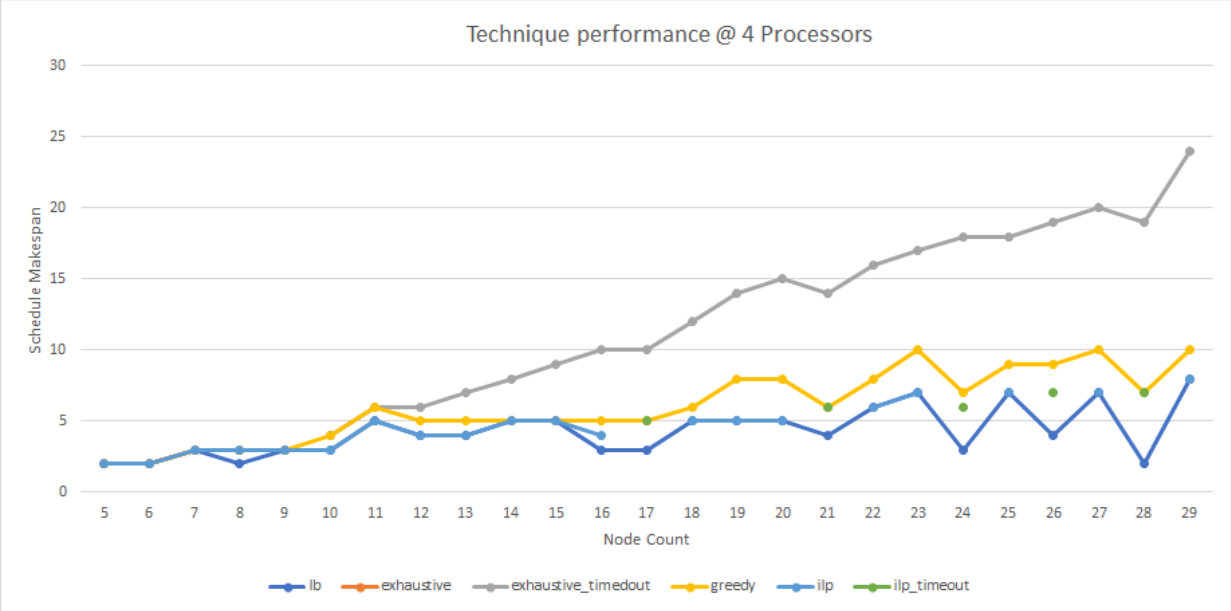

Fig 3. ILP performance with 4 processors against previous techniques @ timeout = 10 seconds with average greedy-lb gap = 1.76

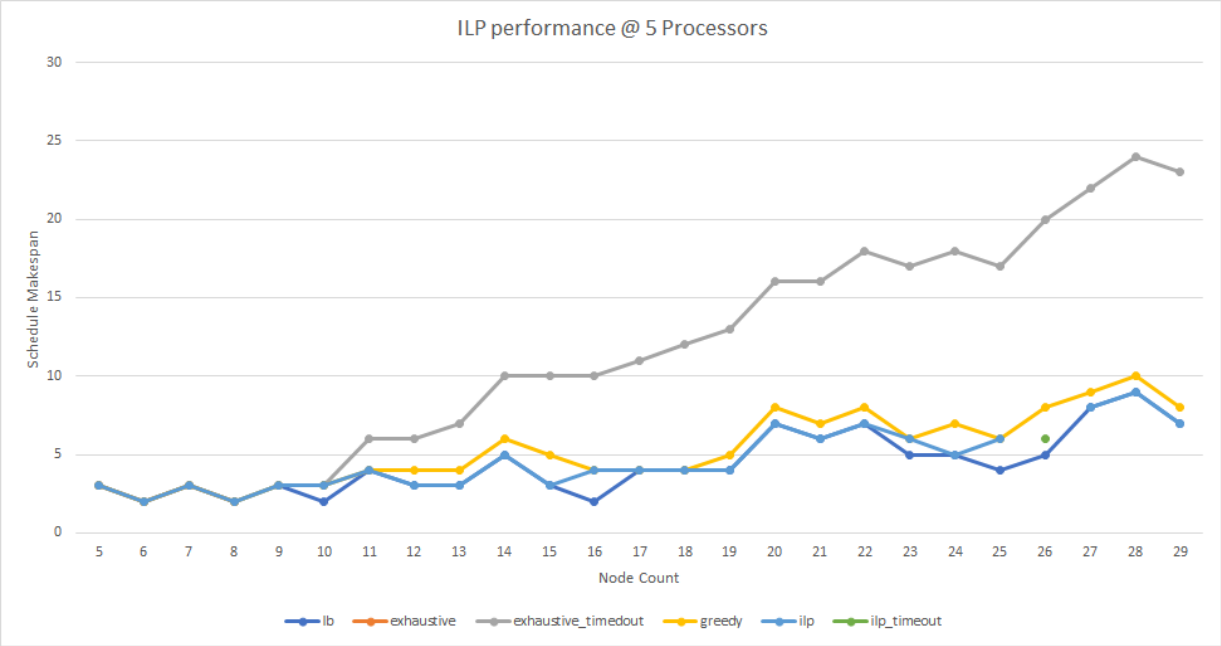

Fig 4. ILP performance with 5 processors against previous techniques @ timeout = 10 seconds with average greedy-lb gap = 0.92

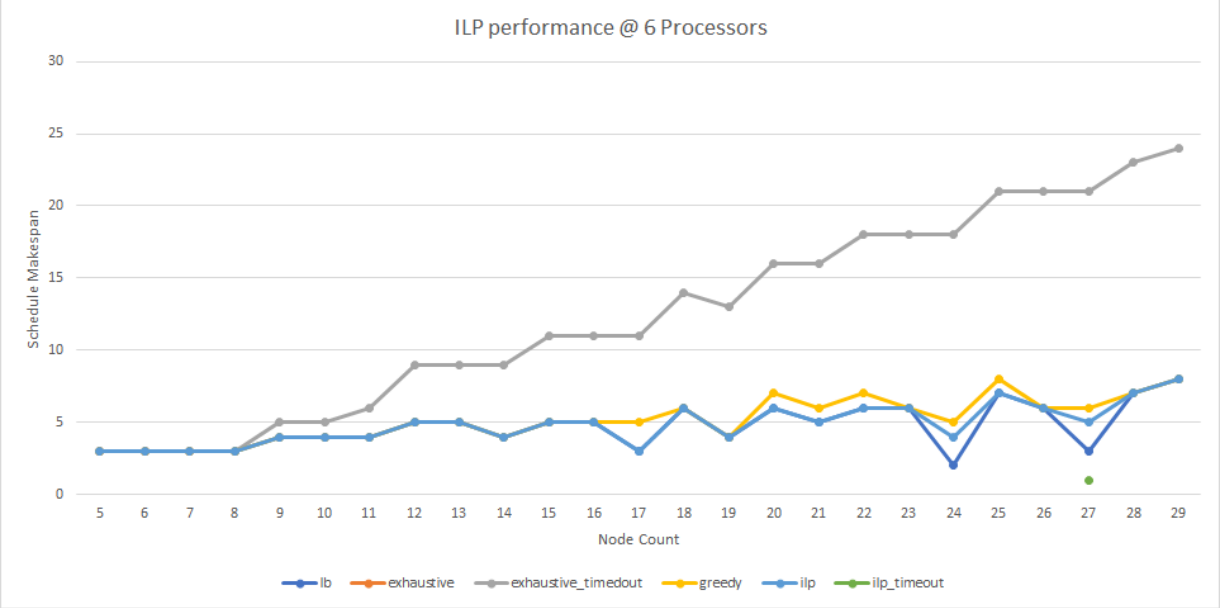

Fig 5. ILP performance with 6 processors against previous techniques @ timeout = 10 seconds with average greedy-lb gap = 0.48

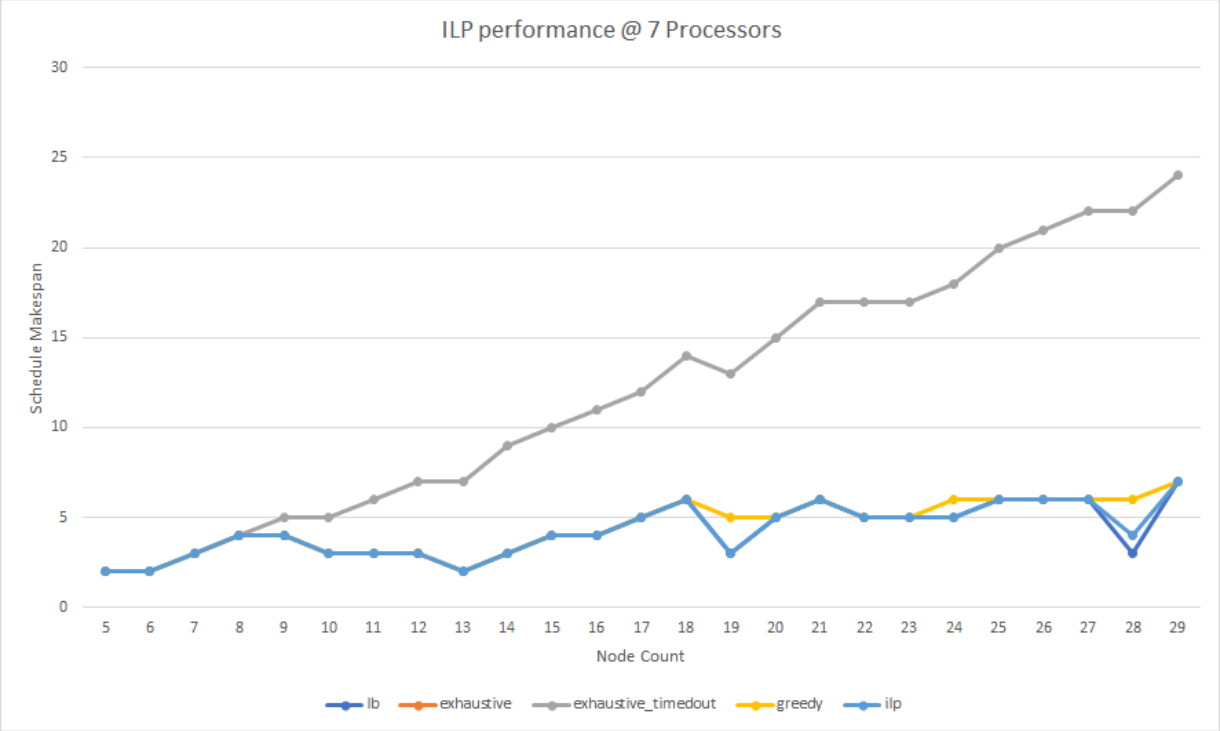

Fig 6. ILP performance with 7 processors against previous techniques @ timeout = 10 seconds with average greedy-lb gap = 0.24

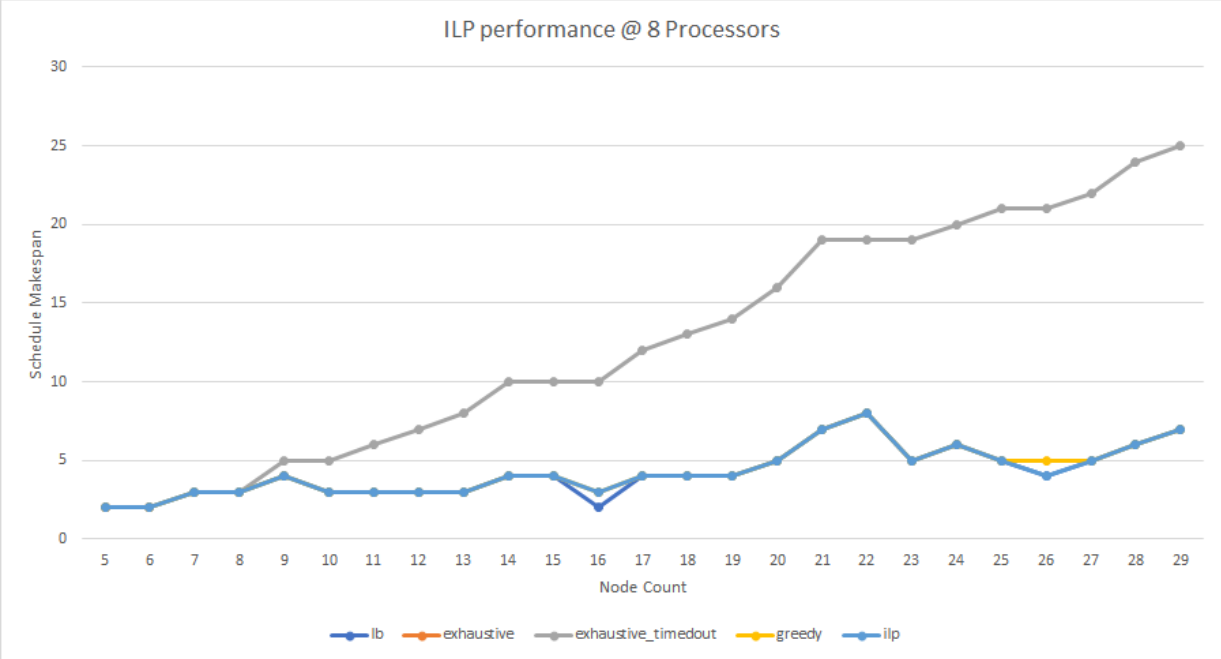

Fig 7. ILP performance with 8 processors against previous techniques @ timeout = 10 seconds with average greedy-lb gap = 0.08

ILP task graph sparsity sensitivity

Fig. 8 shows the effect of task graph sparsity on the solution found by the ILP model. The sparser the graph (as reflected by a lower edge generation probability in layer-by-layer) the less likely the solver can produce any results, however, the denser the graph is the more likely the solver can produce a result. This is understandable given that more edges translate directly to tighter constraints on the schedule which trims the search space. This ignition doesn't always hold though, and a larger timeout may be needed to find optimal/suboptimal results. What is highly likely though is that for a given task graph size there is a minimum sparsity necessary for the solver to produce any result. This was not directly tested for in this project, so this claim is more anecdotal and thus is not based on experimental evidence.

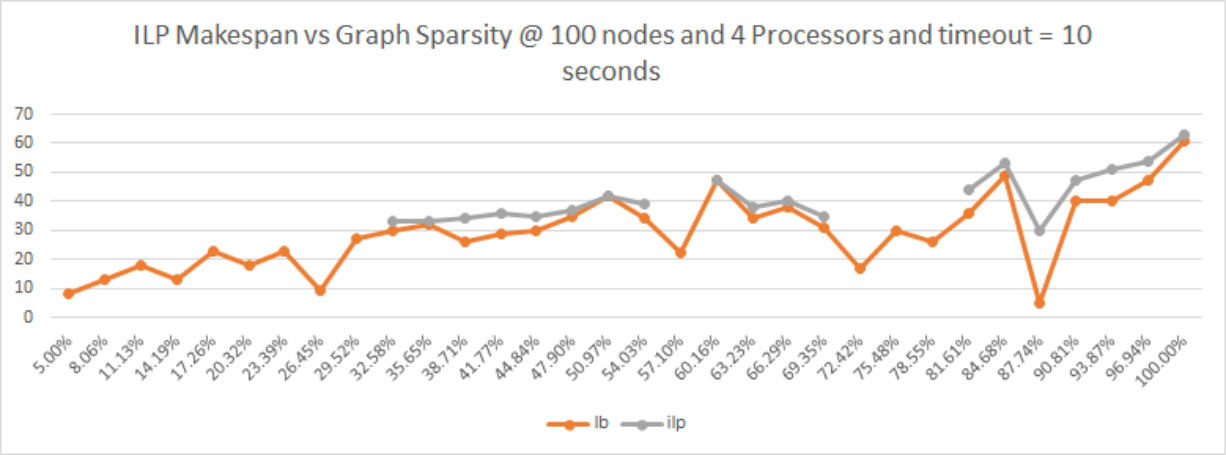

Fig. 8 Task graph sparsity vs ILP makespan

References

[1] Chaudhuri, S., Walker, R.A. and Mitchell, J.E., 1994. Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *2*(4), pp.456-471.

[2] Walkerzy, S.C.R.A., ILP-Based Scheduling with Time and Resource Constraints in High Level Synthesis.

[3] Venugopalan, S. and Sinnen, O., 2014. ILP formulations for optimal task scheduling with communication delays on parallel systems. *Ieee transactions on parallel and distributed systems*, *26*(1), pp.142-151.

[4] Blazewicz, J., Drabowski, M. and Weglarz, J., 1986. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Computer Architecture Letters*, *35*(05), pp.389-393.

[5] Baruah, S., 2020, June. Scheduling DAGs When Processor Assignments Are Specified. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems* (pp. 111-116).

[6] Munier, A. and König, J.C., 1997. A heuristic for a scheduling problem with communication delays. *Operations Research*, *45*(1), pp.145-147.

[7] Campêlo, M., Corrêa, R., Maculan, N. and Protti, F., 2001. Ilp formulations for scheduling ordered tasks on a bounded number of processors. *Electronic Notes in Discrete Mathematics*, *7*, pp.166-169.