Primal Heuristic for the Linear Ordering Problem

Ravi Agrawal[1], Ehsan Iranmanesh[1,2] and Ramesh Krishnamurti[1]
[1]*Simon Fraser University, Burnaby, BC, Canada*
[2]*1QB Information Technologies (1QBit), Vancouver, BC, Canada*

Keywords: Linear Ordering Problem, Linear Programming, Integer Linear Programming, Branch-and-bound, Primal Heuristic, Node Selection.

Abstract: In this paper, we propose a new primal heuristic for the Linear Ordering Problem (LOP) that generates an integer feasible solution from the solution to the LP relaxation at each node of the branch-and-bound search tree. The heuristic first finds a partition of the set of vertices S into an ordered pair of subsets $\{S_1, S_2\}$ such that the difference between the weights of all arcs from S_1 to S_2 and the weights of all arcs from S_2 to S_1 is maximized. It then assumes that all vertices in S_1 precede all vertices in S_2 thus decomposing the original problem instance into subproblems of smaller size i.e. on subsets S_1 and S_2. It recursively does so until the subproblems can be solved quickly using an MIP solver. The solution to the original problem instance is then constructed by concatenating the solutions to the subproblems. The heuristic is used to propose integer feasible solutions for the branch-and-bound algorithm. We also devise an alternate node selection strategy based on the heuristic solutions where we select the node with the best heuristic solution. We report the results of our experiments with the heuristic and the node selection strategy based on the heuristic.

1 INTRODUCTION

The Linear Ordering Problem can be defined as follows. Let $G = (V,A)$ denote a complete digraph on n vertices, where $V = \{1,2,\ldots,n\}$ is the set of vertices and A is the set of arcs that includes, (i,j) and (j,i), for every distinct pair of vertices i and j in V. For every arc $(i,j) \in A$, we have a weight c_{ij}. A *linear ordering* of the vertices $\{1,2,\ldots,n\}$ can be denoted by $\langle v_1, v_2, \ldots, v_n \rangle$, where v_1 precedes v_2, v_2 precedes v_3, and so on. The objective of the problem is to find a linear ordering σ such that $\sum_{i,j:\sigma(i)\prec\sigma(j)} c_{ij}$ is maximized, where $\sigma(i) \prec \sigma(j)$ denotes that vertex i precedes vertex j in the linear ordering σ. The problem belongs to the NP-hard class of problems (Garey and Johnson, 1979).

The Linear Ordering Problem (LOP) is closely related to problems in graph theory, such as the *feedback arc set problem*, and the analogous *feedback node set problem*, as well as the *node induced acyclic subdigraph problem*. The problem has found its relevance in voting theory that involves ranking a set of objects based on individual preferences, such that the ranking matches the individual preferences as closely as possible. This is also relevant to the problem of ranking a set of teams/players in sports tournaments. The *input-output analysis* in the field of economics and the problem of machine scheduling under precedence constraints are other problems of practical importance which can be modeled using the linear ordering problem. For a detailed discussion on the applications of the linear ordering problem, the reader is referred to (Martí and Reinelt, 2011).

As LOP can be used to model problems of such practical importance, a lot of attention has focused on methods to obtain optimal or near optimal solutions to the problem. A common and effective method to solve such problems optimally is Integer Linear Programming. The LOP can be formulated as a 0/1 Integer Linear program and solved using the branch-and-bound approach (Mitchell, 1997). However as the computation time required to obtain the optimal solution grows rapidly with the size of the problem, it becomes impractical for large-sized problems. This makes it necessary to devise techniques for efficiently exploring the branch-and-bound search tree in order to solve large problems using this method. Primal heuristics are often employed to get a good bound early in the branch-and-bound search by generating good integer feasible solutions. The variable and node selection strategies are integral to branch-and-bound as they dictate how the search tree is constructed and

Agrawal, R., Iranmanesh, E. and Krishnamurti, R.
Primal Heuristic for the Linear Ordering Problem.
DOI: 10.5220/0007406301510156
In *Proceedings of the 8th International Conference on Operations Research and Enterprise Systems (ICORES 2019)*, pages 151-156
ISBN: 978-989-758-352-0

explored. We propose a new primal heuristic for the LOP that can be used to provide the bound for the branch-and-bound algorithm as well as define a new node selection strategy.

Numerous techniques have been developed to obtain really good near optimal solutions to such problems. The Local Search Algorithm for the Traveling Salesman Problem (Lin, 1965), for example, starts with some initial feasible solution and then iteratively improves the solution by searching solutions in its neighbourhood. The quality of solutions obtained using such local search methods depends on the size of the neighbourhoods, hence techniques to investigate richer neighbourhoods have been developed. For a detailed survey of such techniques, the reader is referred to (Ahuja et al., 2002). Population based methods such as the Genetic Algorithm (GA), Scatter Search (SS), etc. are well established meta-heuristics which search for a good solution by evolving a set of solutions. These methods have been successfully applied to the Linear Ordering Problem and other similar problems. For a detailed treatment of these methods, please refer (Martí and Reinelt, 2011). Memetic Algorithm (MA) which combines the Genetic Algorithm and local search procedure has also been developed for the Linear Ordering Problem (Schiavinotto and Stützle, 2004). Such methods are capable of finding really good solutions, however, the methods do not guarantee the quality of the solution. They also often take a long time to converge making them impractical to be used as primal heuristics as a part of the branch-and-bound.

Hybrid methods which combine local search and exact methods using ILP techniques have also been proposed (Dumitrescu and Stützle, 2003). In such methods, an instance of the problem is solved by local search methods, while the subproblems are solved optimally, both to explore the neighbourhood of a feasible solution, as well as to obtain good bounds on the optimal solution. A Mixed Integer Program heuristic has been developed for the Linear Ordering Problem (Iranmanesh and Krishnamurti, 2016). This MIP heuristic generates a starting feasible solution based on the Linear Programming solution to the Integer Program formulation for the LOP. For each starting solution, a neighborhood is defined, again based on the LP solution to the problem. A MIP solver is then used to obtain the optimal solution among all the solutions in the neighborhood. As compared to the MIP heuristic, our heuristic relies on partitioning the set of vertices S into an ordered pair of subsets $\{S_1, S_2\}$ such that the difference between the weights of all arcs from S_1 to S_2 and the weights of all arcs from S_2 to S_1 is maximized. The set of vertices are recursively partitioned until for each of the resulting subsets we can quickly solve the linear ordering problem on the subset using a MIP solver. The integer feasible solution to the original LOP instance is then constructed by concatenating the solutions to the linear ordering problems on the subset of vertices. In comparison with MIP heuristic, our heuristic is fast and generates good solutions close to the optimal and hence can be used as a primal heuristic in branch-and-bound.

2 PROBLEM FORMULATION

A linear ordering problem on graph $G = (V,A)$ with arc weights $c_{ij}\ \forall\ (i,j) \in A$, can be formulated as a 0/1 integer programming problem. We define a binary decision variable x_{ij} for each arc $(i,j) \in A$ such that:

$$x_{ij} = \begin{cases} 1, & \text{if } i \prec j, \text{ vertex } i \text{ precedes vertex } j \\ 0, & \text{otherwise} \end{cases}$$

The canonical Integer Linear Programming formulation for the LOP (Martí and Reinelt, 2011) can be given as follows:

$$\text{Maximize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

s.t.

$$x_{ij} + x_{ji} = 1 \qquad \forall\, i,j \in V : i < j \tag{2}$$

$$x_{ij} + x_{jk} + x_{ki} \leqslant 2 \qquad \forall\, i,j,k \in V : i < j, i < k, j \neq k \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i,j \in V : i \neq j \tag{4}$$

The objective function (1) maximizes the total weight of all arcs (i,j) such that $i \prec j$. Constraint (2) ensures that either $i \prec j$ or $j \prec i$, but not both. Constraint (3) prohibits a directed cycle where $i \prec j$, $j \prec k$, and $k \prec i$. Constraint (4) constrains the variables x_{ij} to take values in the set $\{0,1\}$.

3 METHOD

We design the primal heuristic for the linear ordering problem based on the concept of *strongly connected components* in graph theory. A directed graph is said to be *strongly connected* if every vertex in the graph can be reached from every other vertex. The *strongly connected components* of an arbitrary directed graph form a partition into subgraphs that are themselves

Algorithm 1: Partitioning Problem for the Linear Ordering Problem.

Input : Weight matrix $[C]_{n\times n}$, and LP relaxation solution $[P]_{n\times n}$
Output: Subsets (S_1, S_2) of the set of vertices V in graph $G = (V, A)$
for *each* $x_{ij} == 1$ *in* P **do**
  add constraint $(y_i - y_j \geq 0)$ to LOP-Partition formulation;
end
Solve the LOP-Partition (with the added constraints);
Let S_1 be the set of all vertices with $y_i == 1$;
Let S_2 be the set of all vertices with $y_i == 0$;
Return (S_1, S_2);

strongly connected. If every strongly connected component in the graph is contracted to a single vertex, the resulting graph is a *directed acyclic graph*. The linear ordering problem itself is based on a complete directed graph which is strongly connected. However, if we had some way of determining edges at least some of which participate in the optimal solution, we can compute the strongly connected components over this new graph. The resulting directed acyclic graph can then be used to establish the precedence relation between the vertices that belong to the different strongly connected components. The linear ordering problem, as a result, can be decomposed into subproblems on each of the strongly connected components.

To see how this could be useful, consider the graph $G = (V, A)$ for the linear ordering problem. We form a new graph H by greedily selecting the edges (i, j) between each pair of vertices i and j, such that $c_{ij} > c_{ji}$. The strongly connected components of graph H form an acyclic graph $S = (V_s, A_s)$ where $V_s = \{s_1, s_2, \ldots, s_k\}$ is the set of components and A_s is the set of arcs between the components. We can then define the precedence relation between the different components as $s_i \prec s_j$ if $(s_i, s_j) \in A_s$. Consequently, we can also define the precedence relation between the vertices belonging to the different components as $u \prec v$ if $u \in s_i, v \in s_j$ and $s_i \prec s_j$. Once these precedence relations are established, we just need to find the precedence relations between the vertices within each component to solve the complete linear ordering problem.

In practice, however, it is too much to expect that the graph resulting from the greedy selection of edges would contain more than one component. So we devise an ILP problem to find the best partition to divide the linear ordering problem into subproblems.

3.1 Partitioning Problem

For a linear ordering problem, the *partitioning problem* splits the set of vertices V into two subsets S_1 and S_2 such that $\sum_{i\in S_1, j\in S_2}(c_{ij} - c_{ji})$ is maximized. It is formulated as a 0/1 integer programming problem. We define a decision variable y_i for each vertex $i \in V$ such that:

$$y_i = \begin{cases} 1, & \text{if vertex } i \text{ belongs to } S_1 \\ 0, & \text{if vertex } i \text{ belongs to } S_2 \end{cases}$$

The Integer Linear Programming formulation for the partitioning problem can then be given as:

$$\text{Maximize} \sum_{(i,j)\in A} c_{ij}(y_i - y_j) \tag{5}$$

s.t.

$$y_i \in \{0, 1\} \qquad \forall\, i \in V \tag{6}$$

We call this problem LOP-Partition. The objective function (5) maximizes the difference between the total weight of the arcs from subset S_1 to S_2 and the total weight of the arcs from S_2 to S_1. The only set of constraints for the problem, Constraint (6), states that the decision variables take values from the set $\{0, 1\}$.

We can see that the problem is trivial when we have no constraints. However, when using branch-and-bound to solve the LOP, we have the solution to the LP relaxation at each node where some variables x_{ij} have been fixed to either 0 or 1. For all variables x_{ij} set to 1, we introduce additional constraints in the above problem to ensure that we adhere to the partial solution. The constraint can be given as follows:

$$y_i - y_j \geq 0 \tag{7}$$

3.2 Algorithm

The heuristic recursively partitions the set of vertices in the graph $G = (V, A)$ into two subsets. The partitioning is performed until the subsets can be efficiently solved using an MIP solver. If the subsets are sufficiently small, we use the MIP solver to solve them optimally. The pseudocode for the heuristic is given in Algorithm 2.

Algorithm 2: Primal Heuristic to obtain Integer Feasible Solution.

Input : Weight matrix $[C]_{n\times n}$, and LP relaxation solution $[P]_{n\times n}$
Output: An ordering $\langle v_1, v_2, \ldots, v_n\rangle$ of the vertices in graph $G = (V, A)$
if $n \leq 40$ **then**
  return optimal ordering v^* using a MIP solver;
end
Run Algorithm 1 (partitioning problem) to obtain subsets S_1 and S_2;
Find the weight matrix $[C_{S_1}]_{|S_1|\times|S_1|}$ for subset S_1;
Find the weight matrix $[C_{S_2}]_{|S_2|\times|S_2|}$ for subset S_2;
Find the LP solution matrix $[P_{S_1}]_{|S_1|\times|S_1|}$ for subset S_1;
Find the LP solution matrix $[P_{S_2}]_{|S_2|\times|S_2|}$ for subset S_2;
Run Algorithm 2 with $[C_{S_1}]$ and $[P_{S_1}]$ to get ordering v^1 of S_1;
Run Algorithm 2 with $[C_{S_2}]$ and $[P_{S_2}]$ to get ordering v^2 of S_2;
Obtain v by concatenating v^1 and v^2;
Return v;

3.3 Improvement Phase

The linear ordering $\langle v_1, v_2, \ldots, v_n\rangle$ obtained via the primal heuristic as described in Algorithm 2 serves as a starting solution for improvement heuristics. We use local search to look for *insert* moves that further improve the objective function (1) (Laguna et al., 1999). We perform the move that causes maximum improvement until no improvement is possible using such a move.

3.4 Node Selection

When using the branch-and-bound search in ILP, we may have multiple nodes in the queue waiting to be processed. For each node, we have the solution to the LP relaxation and the branching variable at its parent node, and the branching direction. This information is used to provide the partial solution to the heuristic and a node with the best heuristic solution is chosen to be processed next.

4 EMPIRICAL ANALYSIS

The computational experiments were conducted on an Intel Core i7-7700HQ with 2.80 GHz 64-bit processor, 8.0 GB of RAM and Ubuntu 16.04 64-bit as the Operating System. The heuristic was implemented in C++. The LP and MIP problems were solved using CPLEX 12.8 on a single thread. The experiments on the difficult instances were run under the time restriction of 600 seconds however no such constraint was placed on the easier instances.

4.1 Data Set

The data set for the computational experiments was obtained from the Optsicom project (Martí et al., 2009). We use the following data for our experiments.

4.1.1 Special Instances

These are problem instances that were used in publications. The *EX* instances were used in (Christof and Reinelt, 2001). The *ATP* instances were created from the results of ATP tennis tournaments in 1993/1994. Nodes correspond to a selection of players and the weight of an arc (i, j) is the number of victories of player i against player j.

4.1.2 Instances RandomAI

These instances are generated from a uniform distribution in the range $[0, 100]$. These problems were originally generated from a [0, 25000] uniform distribution (Laguna et al., 1999) and modified afterwards, sampling from a significatively narrow range ([0,100]) to make them harder to solve. The size of these instances are 100, 150, and 200. For the experiments, we use the instances with size 100 as the larger instances are too difficult to be solved using ILP.

4.2 Computational Results

We summarize the computational results in Table 1 and Table 2. Table 1 shows experiments on special instances that can be solved in reasonable time using ILP, and Table 2 shows experiments on the much harder RandomAI instances. In Table 1, we can see that using the heuristic (H), and using the heuristic

Table 1: Computational Results for Special Instances.

		CPLEX		H		H+NS	
Instance	Size	NLPS	CPU(s)	NLPS	CPU(s)	NLPS	CPU(s)
ATP66	66	113	56	57	41	62	44
ATP76	76	1046	1147	651	778	254	368
EX4	50	133	22	97	21	128	46
EX5	50	197	35	183	45	123	42
EX6	50	47	10	37	12	58	18

Table 2: Computational Results for RandomAI Instances.

		Best Known (Martí et al., 2009)			H		H+NS	
Instance	Size	Solution	Bound	Gap(%)	Solution	Gap(%)	Solution	Gap(%)
t1d100.01	100	106852	114468	7.128	106288	7.696	106344	7.639
t1d100.02	100	105947	114077	7.674	104905	8.743	104952	8.694
t1d100.03	100	109819	117843	7.306	109035	8.078	109184	7.931
t1d100.04	100	109252	117639	7.677	108417	8.506	108675	8.248
t1d100.05	100	108859	117538	7.973	108094	8.737	108447	8.383
t1d100.06	100	108201	117057	8.185	107786	8.601	107609	8.779
t1d100.07	100	108803	117118	7.642	108276	8.166	108405	8.037
t1d100.08	100	107480	115756	7.700	106746	8.440	107010	8.173
t1d100.09	100	108549	116527	7.350	107720	8.176	107776	8.120
t1d100.10	100	108771	117518	8.042	108222	8.590	108336	8.475

along with node selection (H+NS), both lead to a decline in the number of LPs (NLPS) solved. However, for some instances we can see an increase in the CPU time (in seconds) which may be due to the additional work at each node. Table 2 shows results for the much harder instances for which we report the lower bounds obtained. The heuristic can be used to find solutions which are substantially close to the best known solutions (Martí et al., 2009) thereby making it possible to prune large parts of the tree quickly.

5 CONCLUSION

The Linear Ordering Problem is a classic combinatorial optimization problem with applications to numerous problems of practical importance. A standard method to solve such problems optimally is using Integer Linear Programming and branch-and-bound search. The computation time, however, grows rapidly with the size of the problem instance. In this paper, we propose a new primal heuristic that generates good feasible solutions from the partial LP solution at each node of the branch-and-bound tree. We also devise a new node selection strategy based on the heuristic solution. Preliminary experimental results show that the approach is promising. The solutions obtained using the heuristic are substantially close to the optimal and provide a good lower bound for the branch-and-bound algorithm. The number of nodes that need to be processed also show a decline when the heuristic is used.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive comments and suggestions. The first and last author would like to acknowledge support from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

Ahuja, R. K., Ergun, O., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood

search techniques. *Discrete Applied Mathematics*, 123(1-3):75 – 102.

Christof, T. and Reinelt, G. (2001). Algorithmic aspects of using small instance relaxations in parallel branch-and-cut. *Algorithmica*, 30(4):597–629.

Dumitrescu, I. and Stützle, T. (2003). Combinations of local search and exact algorithms. In *Applications of Evolutionary Computation*, pages 211–223. Springer.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Iranmanesh, E. and Krishnamurti, R. (2016). Mixed integer program heuristic for linear ordering problem. In *5th International Conference on Operations Research and Enterprise Systems*, pages 152–156.

Laguna, M., Martí, R., and Campos, V. (1999). Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & Operations Research*, 26(12):1217 – 1230.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269.

Martí, R. and Reinelt, G. (2011). *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Applied Mathematical Sciences Volume 175. Springer.

Martí, R., Reinelt, G., and Duarte, A. (2009). Optsicom project. *http: www.optsicom.es/lolib*.

Mitchell, J. E. (1997). Solving linear ordering problems with a combined interior point simplex cutting plane algorithm.

Schiavinotto, T. and Stützle, T. (2004). The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402.