Mixed Integer Program Heuristic for Linear Ordering Problem

Ehsan Iranmanesh and Ramesh Krishnamurti

School of Computing Science, Simon Fraser University, Burnaby, Canada

Keywords: Linear Ordering Problem, Linear Programming, Integer Linear Program, Mixed Integer Program Heuristic.

Abstract: The Linear Ordering Problem is a classic optimization problem which can be used to model problems in graph theory, machine scheduling, and voting theory, many of which have practical applications. Relatively recently, there has been some success in using Mixed Integer Program (MIP) heuristic for NP-hard optimization problems. We report our experience with using a MIP heuristic for the problem. Our heuristic generates a starting feasible solution based on the Linear Programming solution to the IP formulation for the Linear Ordering Problem. 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. The MIP heuristic shows promise for large problems of hard instances.

1 INTRODUCTION

In the Linear Ordering Problem, we are given a directed graph $G = (V,A)$ with nodes $V = \{1,2,\ldots,n\}$ and two directed arcs, (i,j) and (j,i), between every pair of nodes i and j. Each arc $(i,j) \in A$ has weight c_{ij}. Let $< v_1, v_2, \ldots, v_n >$ denote a linear ordering of the nodes $V = \{1,2,\ldots,n\}$, where v_1 precedes v_2, v_2 precedes v_3, and so on, in this ordering. We seek 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 $\sigma(i)$ precedes $\sigma(j)$ in the linear ordering σ. This problem has been shown to be NP-Hard (Garey and Johnson, 1979).

The Linear Ordering Problem (LOP) can be used to model problems in graph theory, such as the feedback arc/node set problem and the node induced acyclic sub digraph problem. The linear ordering problem can also be used to model problems where we need a rank ordering of certain objects using aggregation of individual preferences, so that the ranking we derive matches the individual preferences as closely as possible. This has applications in voting theory, as well as deriving a ranking of players/teams in sports tournaments, to name a few. The linear ordering problem also has applications in machine scheduling, where a set of jobs with precedence constraints need to be scheduled on a single machine. For a detailed treatment of the applications of the linear ordering problem, see (Marti and Reinelt, 2011).

Since it can be used to model many problems of practical importance, a great deal of attention has focused on techniques to obtain either optimal or near optimal solutions to the linear ordering problem. A standard method to solve such a problem optimally is to first obtain an Integer Linear Programming (ILP) formulation for the problem, and then obtain an optimal integer solution, using established methods such as branch and bound. The bounds used in such a branch and bound method may be either a Linear Programming Relaxation bound or a Lagrangian Relaxation bound. However, the computational time required to obtain the optimal solution using such methods grows rapidly with the size of the problem, and these methods become impractical for large-sized problems.

Algorithms/methods to obtain near optimal solutions to these problems have also been developed. An early example of using such an approach with great success was the Local Search Algorithm for the Traveling Salesman Problem (Lin, 1965). The local search algorithm starts with some initial feasible solution and then iteratively improves the current solution by searching solutions in its neighborhood. Since the quality of the solutions obtained using a local search algorithm often depends on the size of the neighborhoods, techniques to investigate richer neighborhoods have been developed. For a comprehensive survey of such techniques, see (Ahuja et al., 2002). In practice, local search algorithms have been shown to yield the best performance on large instances of computationally hard problems.

More recently, methods have evolved which use a combination of local search methods and exact methods using ILP techniques (Dumitrescu and Stutzle,

Iranmanesh, E. and Krishnamurti, R.
Mixed Integer Program Heuristic for Linear Ordering Problem.
DOI: 10.5220/0005710701520156
In *Proceedings of 5th the International Conference on Operations Research and Enterprise Systems (ICORES 2016)*, pages 152-156
ISBN: 978-989-758-171-7

2003). In such hybrid methods, an instance of the problem is solved by local search methods, while subproblems are solved optimally, both to explore the neighborhood of a feasible solution, as well as obtain good bounds on the optimal solution. Lourenco first explored hybrid methods (Loureno, 1995), where an iterated local search is used to solve the job-shop problem, and ILP techniques are used to optimally solve subproblems to perturb a locally optimal solution. Applegate et al. (Applegate et al., 1999) extend this approach for the Traveling Salesman Problem (TSP) by first running iterated local search multiple times on a TSP instance, and retain the best locally optimal tour each time. Then they construct a restricted graph which has the same set of nodes as the original graph, but with edges that appear at least once in the tours retained. Finally, they solve the TSP optimally on the restricted graph using ILP techniques. to obtain the optimal solution for the original TSP instance.

In a different approach using local search to solve the TSP, Burke et al. (Burke et al., 2000; Burke et al., 2001) employ ILP techniques to optimally solve a subproblem, whose solution corresponds to the optimal solution among a large number of solutions in the neighborhood of a feasible solution. Maniezzo et al. (Maniezzo, 1999) uses an exact method to solve the Quadratic Assignment Problem, with an approximate heuristic method, an approximate non-deterministic tree search in an ant colony optimization algorithm, to obtain a good bound on the optimal solution.

In related work, Mixed Integer Programming (MIP) heuristics use Local Branching (LB), Variable neighborhood Branching (VNB), or Variable neighborhood Decomposition Search (VNDS-MIP), to solve $0-1$ Mixed Integer Linear Programming (MILP) problems. Fischetti et al. (Fischetti and Lodi, 2003) use LB, which starts with a feasible solution to the $0-1$ MILP P. Constraints are then added to P that exclude all solutions with hamming distance greater than or equal to k, for some fixed k. The problem thus derived is then solved optimally using a MIP solver. Local search is used to find the optimal value for k.

2 PROBLEM FORMULATIONS

The linear ordering problem may be formulated as an integer linear program. For each pair of nodes i, j, x_{ij} is defined as follows:

$$x_{ij} = \begin{cases} 1, & \text{if } i \prec j \text{ (node } i \text{ precedes node } j\text{)} \\ & \text{in the ordering} \\ 0, & \text{otherwise} \end{cases}$$

The weight of arc (i, j) in the given instance is denoted c_{ij}. The ILP formulation for LOP is given below (Marti and Reinelt, 2011). We refer to this as Problem LOP:

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

$$\text{s.t.} \tag{2}$$

$$x_{ij} + x_{ji} = 1 \qquad \forall i, j \in V \tag{3}$$

$$x_{ij} + x_{jk} + x_{ki} \leqslant 2 \qquad \forall i, j, k \in V \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{5}$$

The objective function (1) maximizes the total weights. The Constraint (3) ensures that either $i \prec j$ or $j \prec i$, but not both. The constraint (4) prohibits a cycle where $i \prec j$, $j \prec k$, and $k \prec i$. The constraint (5) constrains the variables x_{ij} to take values in the set $\{0, 1\}$.

3 METHODS USED

Both the heuristics we use for this problem in this paper are based on a MIP solver. Each heuristic fixes the value (assigns a value of either 0 or 1) for each variable in a subset of the variables, and keeps the remaining variables free. The subproblem that needs to be solved to determine the values the free variables assume is then solved optimally using the MIP solver. The only difference between the heuristics is in the method they use to determine the subset of variables whose values are fixed.

Algorithm 3 below outlines the MIP heuristic we use to solve the LOP. We first derive a feasible integer solution to the problem from an optimal LP solution to LOP. We call such a solution a *starting* solution. Any feasible integer solution, including the starting solution, specifies an ordering of the nodes. Let the starting solution, denoted v^s, specify the ordering $< v_1^s, v_2^s, \ldots, v_n^s >$. Given a starting solution v^s to the LOP, and a set N of neighborhood solutions feasible to the LOP, we obtain the optimal solution in set N using the MIP solver CPLEX (Algorithm 2). We use the solution thus obtained as our new starting solution, and repeat this procedure for a number of iterations. We return the best feasible solution obtained.

Algorithm 1 below specifies the steps used to obtain a starting solution from the optimal LP solution to the LOP. Let $\hat{x} = \hat{x}_{ij}, (i, j) \in A$ denote the optimal LP solution. If $n \leq 50$, a starting solution x^s is obtained by solving the LOP optimally, with additional constraints $x_{ij} = 1$ if $\hat{x}_{ij} \geq 0.9$, and $x_{ij} = 0$ if $\hat{x}_{ij} \leq 0.1$. The sequence obtained is the starting solution. If $n \geq 50$, the LOP is solved approximately, with additional constraints $x_{ij} = 1$ if $\hat{x}_{ij} \geq 0.9$, and $x_{ij} = 0$

if $\hat{x}_{ij} \leq 0.1$. Let $v^i =< v^i_1, v^i_2, \ldots, v^i_n >$ denote the intermediate ordering thus obtained. If $k = \lfloor n/2 \rfloor$, we form two subproblems, P1 with nodes $\{v^i_1, v^i_2, \ldots, v^i_k\}$, and P2 with nodes $\{v^i_{k+1}, v^i_{k+2} \ldots, v^i_n\}$. We solve subproblems P1 and P2 optimally using CPLEX, to obtain subsequences $v^1 =< v^1_1, v^1_2 \ldots, v^1_k >$ and $v^2 =< v^2_{k+1}, v^2_{k+2}, \ldots, v^2_n >$. The two subsequences are then concatenated to obtain the starting sequence $v^s =< v^1_1, v^1_2, \ldots, v^1_k, v^2_{k+1}, \ldots, v^2_n >$. For instances with at least 200 objects, solving the LP is time consuming. Therefore we use a random permutation of the nodes as a starting solution.

We describe below the set N of neighboring solutions to feasible solution $v =< v_1, v_2, \ldots, v_n >$. The solution v can be also considered as a sequence of the integers v_1, v_2, $\ldots$, v_n, where $< v_1, v_2, \ldots, v_n >$ is a permutation of the integers $< 1, 2, \ldots, n >$. Given such a sequence, we can define a subsequence $v_{i,j} =< v_i, v_{i+1}, \ldots, v_j >$, where $1 \leq i \leq j \leq n$. The neighborhood N_{ij} (with respect to the subsequence $v_{i,j}$) is defined as the set of all orderings $w =< w_1, w_2, \ldots, w_n >$, where $w_i = v_i$ for $1 \leq i \leq i-1$, and $w_i = v_i$ for $j+1 \leq i \leq n$. Note that in the ordering w, the subsequence $w_{1,i-1} =< w_1, w_2, \ldots, w_{i-1} >$ is identical to the subsequence $v_{1,i-1} =< v_1, v_2, \ldots, v_{i-1} >$, and the subsequence $w_{j+1,n} =< w_{j+1}, w_{j+2}, \ldots, w_n >$ is identical to the subsequence $v_{j+1,n} =< v_{j+1}, v_{j+2}, \ldots, v_n >$. However, the subsequence $w_{i,j} =< w_i, w_{i+1}, \ldots, w_j >$ can be any reordering of the subsequence $v_{i,j} =< v_i, v_{i+1}, \ldots, v_j >$. In addition, for every pair $k, l, 1 \leq k \leq i-1, i+1 \leq l \leq j$, $w_k \prec w_l$, and for every pair $p, q, i \leq p \leq j, j+1 \leq q \leq n$, $w_p \prec w_q$. In other words, every node in the subsequence $w_{1,i-1}$ precedes every node in the subsequence $w_{i,j}$, and every node in the subsequence $w_{i,j}$ precedes every node in the subsequence $w_{j+1,n}$. Thus the size of the neighborhood set N_{ij} is $(j-i+1)!$, which is exponential in the size of the subsequence $s = (j-i+1)$. The subsequence itself is specified in terms of the start position i, and the size of the subsequence, s.

4 EMPIRICAL ANALYSIS

The computational experiments were conducted on an Intel Core i7 with 2.8 GHz 64-bit processor, 16.0 GB of RAM and OSX Yosemite 64-bit as the Operating System. The heuristic was implemented in C++. The LP and MILP problems were solved with CPLEX 12.5. All the experiments ran under the time limitation of 500 seconds.

Algorithm 1: LP Based Heuristic to obtain Starting Solution.

Input : Cost Matrix $[C]_{n \times n}$
Output: An ordering $v =< v_1, v_2, \ldots, v_n >$ of the nodes in the graph $G = (V, A)$

1 Solve LP relaxation of the problem and get the $\hat{x} = \{x_{ij}\}_{i,j \in N}$;
2 $\forall i, j$, set the variable $x_{ij} = 1$ where $\hat{x}_{ij} >= 0.9$;
3 $\forall i, j$, set the variable $x_{ij} = 0$ where $\hat{x}_{ij} <= 0.1$;
4 Solve LOP optimally with the additional constraints 2 and 3 above;
5 **if** $n \leq 50$ **then**
6   return optimal ordering v^s;
7 **end**
8 **if** $n \geq 50$ **then**
9   Solve LOP approximately with MIP solver;
10   Obtain the intermediate ordering $v^i =< v^i_1, v^i_2, \ldots, v^i_n >$;
11   Generate two subproblems P1 with nodes $\{v^i_1, v^i_2, \ldots v^i_k\}$ and P2 with nodes $\{v^i_{k+1}, v^i_{k+2}, \ldots, v^i_n\}$ (where $k = \lfloor n/2 \rfloor$);
12   Solve each subproblem optimally by CPLEX and get subsequences $v^1 =< v^1_1, \ldots, v^1_k >$, $v^2 =< v^2_{k+1}, \ldots, v^2_n >$;
13   Concatenate the two subsequences to return sequence $v^s =< v^s_1, \ldots, v^s_k, v^s_{k+1}, \ldots, v^s_n >$;
14 **end**

4.1 Data Set

Instances RandAI: These instances are generated from a uniform distribution in the range $[0, 100]$. They were initially generated from a uniform distribution in the range $[0, 25000]$, and then sampled from the much narrower range $[0, 100]$. These are the harder instances. The number of nodes in these instances are 100, 150, and 200.

Instances RandAII: These contain easier instances which are generated from randomly generated permutations. Again, the number of nodes in these instances are 150, and 200.

For a detailed treatment of the methods that have been used to solve the Linear Ordering Problem, and a description of these instances, see (Marti et al., 2009).

4.2 Computational Results

We summarize the computational results in Table 1 and Table 2. Table 1 shows experiments on instances of RandAI, and Table 2 shows experiments on instances of RandAII. Table 1 shows that for the harder instances for the sizes 100 and 150, the largest gap obtained by CPLEX is almost 24% where the largest

Algorithm 2: MIP Solver to obtain Optimal Solution in Neighborhood.

Input : An ordering $v = < v_1, v_2, \ldots, v_n >$, and a subsequence $v_{i,j}$

Output: An ordering $\hat{v} = < \hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n >$ of the nodes in the graph $G = (V, A)$

for *all* $s, t \in [1, 2, \ldots, n]$ **do**

    if $s, t \notin [i, \ldots, j]$ *and* $v_s \succ v_t$ **then**

        $x_{s,t} = 1$;

        $x_{t,s} = 0$;

    else if $s \notin [i, \ldots, j]$ *and* $t \in [i, \ldots, j]$ *and* $v_s \succ v_t$ **then**

        $x_{s,t} = 1$;

        $x_{t,s} = 0$;

    else if $s \notin [i, \ldots, j]$ *and* $t \in [i, \ldots, j]$ *and* $v_s \prec v_t$ **then**

        $x_{s,t} = 0$;

        $x_{t,s} = 1$;

end

Solve LOP with the additional constraints specified above;

Return the ordering obtained $\hat{v}$;

Algorithm 3: Mixed Integer Program Heuristic.

Input : Cost Matrix $[C]_{n \times n}$

Output: An ordering $\hat{v} = < v_1, v_2, \ldots, v_n >$ of the nodes in the graph $G = (V, A)$

Run Algorithm I (LP Based Heuristic) and get Starting Solution v^s;

Let $|N|$ denote the size of the neighborhood set;

$N = t_1$ (Starting size for threshold is 5);

while *number of iterations* $i < I$ **do**

    Pick a subsequence $v^s_{i,j}$ of size N randomly from $\hat{v}$;

    Run Algorithm 2 with the inputs: ordering $\hat{v}$, subsequence $\hat{v}_{i,j}$;

    Obtain output ordering $\hat{v}$;

    Update $v^s = \hat{v}$;

    if *there is no improvement after a number of iterations* **then**

        $N = N + \delta$ (δ is 5 for earlier iteration and is 2 for later iterations)

    end

end

gap obtained by the heuristic is 15%. For the easier instances, the largest gap obtained by the CPLEX is 2.14% and the largest gap obtained by the heuristic is 0.85%.

Table 1: Computational results for Instances RandAI.

	CPLEX			MIP Heuristic	
Instance	Sol	Gap %	Best Bound	Sol	Gap%
N-t1d100.01	96637	18.45%	114468	105181	8.82%
N-t1d100.02	97508	16.99%	114077	105438	8.2%
N-t1d100.03	99330	18.64%	117843	108728	8.38%
N-t1d100.04	99457	18.28%	117639	107954	8.97%
N-t1d100.05	99113	18.59%	117538	107582	9.25%
N-t1d100.06	96924	20.77%	117057	107075	9.32%
N-t1d100.07	97703	19.87%	117118	107696	8.74%
N-t1d100.08	97224	19.06%	115756	106051	9.15%
N-t1d150.01	213538	20.97%	258334	228870	12.87%
N-t1d150.02	213433	21.37%	259050	225853	14.6%
N-t1d150.03	211917	23.66%	262076	229910	13.99%
N-t1d150.04	212812	21.99%	259619	225965	14.89%
N-t1d150.05	214151	21.34%	259863	228907	13.52%
N-t1d150.06	212907	22.13%	260035	227477	14.31%
N-t1d150.07	214337	21.73%	260922	230268	13.31%
N-t1d150.08	216617	20.46%	260956	229620	13.64%

Table 2: Computational results for Instances RandAII.

	CPLEX			MIP Heuristic	
Instance	Sol	Gap%	Best Bound	Sol	GAP%
N-t2d150.01	75296	1.3%	76276	76021	0.33%
N-t2d150.02	72889	1.26%	73811	73617	0.26%
N-t2d150.03	68926	1.4%	69894	69703	0.27%
N-t2d150.04	73189	1.29%	74136	73960	0.23%
N-t2d150.05	78890	1.21%	79847	79723	0.15%
N-t2d150.06	74684	1.28%	75604	75438	0.22%
N-t2d150.07	73143	1.26%	74067	73852	0.29%
N-t2d150.08	66586	1.82 %	67803	67463	0.5%
N-t2d200.01	146132	1.48%	148295	147704	0.4%
N-t2d200.02	142566	1.63%	144904	144100	0.55
N-t2d200.03	139632	1.77%	142105	141260	0.5%
N-t2d200.04	149170	1.54%	151471	149520	1.3%
N-t2d200.05	148694	1.45%	150854	150176	0.45%
N-t2d200.06	139592	1.73%	142009	141100	0.64%
N-t2d200.07	148132	1.51%	150379	149126	0.84%
N-t2d200.08	148042	1.6%	150415	149650	0.51%
N-t2d200.09	140190	1.82%	142747	141896	0.5%
N-t2d200.10	147990	1.51%	150232	149612	0.41%
N-t2d200.11	145990	1.55%	148262	147398	0.58%
N-t2d200.12	150794	1.5%	153064	152394	0.43%
N-t2d200.13	135922	1.90%	138514	137572	0.68%
N-t2d200.14	142542	1.78%	145083	144108	0.67%
N-t2d200.16	145680	1.67%	148114	147356	0.51%
N-t2d200.17	129950	2.13%	132728	131850	0.66%
N-t2d200.18	149908	1.17%	151663	150948	0.47%
N-t2d200.19	35572	1.925%	138182	137272	0.66%
N-t2d200.20	145180	1.28%	147051	146448	0.41%
N-t2d200.21	141700	1.77%	144221	143428	0.55%
N-t2d200.22	145238	1.57%	147524	146828	0.47%
N-t2d200.23	143480	1.56%	145731	144932	0.55%
N-t2d200.24	149658	1.41%	151769	151208	0.37%
N-t2d200.25	147476	1.52%	149723	149078	0.43%

5 CONCLUSION

The Linear Ordering Problem is a classic optimization problem with wide applications. Solving the problem optimally for the larger hard instances has proved difficult. In this paper, we report our experience with using a MIP heuristic for the problem. Our heuristic generates a starting feasible solution based on the LP relaxation of the IP formulation for the Linear Ordering Problem. 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. Preliminary results indicate that this approach is promising. The solutions obtained using this heuristic are substantially closer to the optimal than the solutions obtained using the MIP solver to solve the entire problem.

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.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (1999). Finding tours in the traveling salesman problem. Technical report, Forschungsinstitut fur Diskrete Mathematik, University of Bonn, Germany.

Burke, E. K., Cowling, P. I., and Keuthen, R. (2000). Embeded local search and variable neighborhood search heuristics applied to the travelling salesman problem. Technical report, University of Nottingham.

Burke, E. K., Cowling, P. I., and Keuthen, R. (2001). Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, pages 203–212, London, UK, UK. Springer-Verlag.

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

Fischetti, M. and Lodi, A. (2003). Local branching. *Math. Program.*, 98(1-3):23–47.

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

Lin, S. (1965). Computer solutions of the traveling salesman problem. 44(10):2245–2269.

Loureno, H. R. (1995). Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83(2):347 – 364. EURO Summer Institute Combinatorial Optimization.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369.

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

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