SOLVING THE SEQUENTIAL ORDERING PROBLEM WITH AUTOMATICALLY GENERATED LOWER BOUNDS

István T. Hernádvölgyi

February 2004

Technical Report # DIT-04-018

Solving the Sequential Ordering Problem with Automatically Generated Lower Bounds

István T. Hernádvölgyi

Department of Information and Communication Technology, University of Trento, Povo, 38100 Trento, Italy*

Abstract. The Sequential Ordering Problem (SOP) is a version of the Asymmetric Traveling Salesman Problem (ATSP) where precedence constraints on the vertices must also be observed. The SOP has many real life applications and it has proved to be a great challenge (there are SOPs with 40-50 vertices which have not been solved optimally yet with significant computational effort). We use novel branch&bound search algorithms with lower bounds obtained from homomorphic abstractions of the original state space. Our method is asymptotically optimal. In one instance, it has proved a solution value to be optimal for an open problem while it also has matched best known solutions quickly for many unsolved problems from the TSPLIB. Our method of deriving lower bounds is general and applies to other variants of constrained ATSPs as well.

1 Introduction

The Sequential Ordering Problem (SOP) is stated as follows. Given a graph G, with n vertices and directed weighted edges with the start and terminal vertices designated. Find a minimal cost Hamiltonian path from the start vertex to the terminal vertex which also observes precedence constraints. An instance of a SOP can be defined by an $n \times n$ cost matrix C, where the entry $C_{i,j}$ is the cost of the edge ij in G, or it is -1 to represent the constraint that vertex j must precede vertex i in the solution path.

The SOP is a model for many real life applications, ranging from helicopter routing between oil rigs [10] to scheduling on-line stacker cranes in an automated warehouse [1].

Most asymptotically optimal solvers model the SOP as an Integer Program. Unfortunately the exact structure of the SOP polytope is not yet fully understood and therefore these methods achieved only limited success. Our approach is state space search. The partial completions of feasible tours form a directed acyclic graph. The lower bounds are derived from abstractions of the original state space and correspond to optimal tour completion costs in the abstract space. The lower bounds are stored in a look-up table which we will refer to as the *pattern database*. They are named so, because the abstraction corresponds to merging states of the original state space according to some syntactic *pattern*. The abstraction mechanism described in this paper is general and it is applicable to other versions of constrained ATSPs as well.

* work done at SITE, University of Ottawa, Canada, istvan@site.uottawa.ca

2 Related Work

Optimal solutions to some instances of the SOP were obtained by Ascheuer *et. al.* [2] who used the cutting plane technique. They also employed heuristic tour constructions and improvements to derive actual solutions. Escudero et al. [5] used a similar approach but the lower bounds were obtained by Lagrangian relaxation.

The HAS-SOP system of Gambardella et al. [6] is a metaheuristic technique. In many instances they obtained the best known upper bounds to unsolved instances. This approach is a form of stochastic search and therefore optimal solutions cannot be guaranteed, however the solutions were obtained very quickly. Similar results using genetic algorithms were achieved by Seo et al. [11].

Christofides et al. [3] considered state space relaxations first to generate lower bounds for TSPs. They also used a state representation very close to ours. The same approach was also considered by Mingozzi et al. [9] for the TSP with time windows and precedence constraints.

The pattern database technique was invented by Culberson and Schaeffer [4] and was later used by Korf [8] to obtain optimal solutions to the Rubik's Cube for the first time.

3 Lower Bounds

In our representation, a SOP state s corresponds to a partial completion of the tour. It records the current last vertex in this partial tour as well as the vertices which have not been reached yet. This is very similar to the state representation of Christofides et al. [3]. The SOP state space S is a lattice with the start state at the apex and the goal state at the bottom. The lattice has n levels, each corresponding to adding a new vertex to a partial tour such that it still satisfies the constraints. These levels are manifested by the edge structure of the lattice. Edges in S only exist between adjacent levels.

The abstract state space S' is also a lattice with the same number of levels as S. However, $|S'| < |S|$, so we can enumerate it efficiently. The abstract lattice is obtained by clustering states of S on the same level. Figure 1 illustrates the conceptual relationship between the original and abstract lattices. S is the original lattice and S' is the resulting abstraction. We chose states on the same levels to cluster and identified the cheapest cost edges entering and leaving these clusters (drawn with bold edges). These edges will be retained to connect the clustered states which are now replaced by a single vertex (shown as filled circles) in S'. The cost of the optimal completion of s' in S' is a lower bound on the optimal completion cost of the preimage states of s' in S. This is exactly what our lower bounds correspond to. The abstractions are simple relabeling functions we call *domain abstractions.* Initially we assign a unique label to each vertex in the SOP. 0 and $n-1$ are the labels of

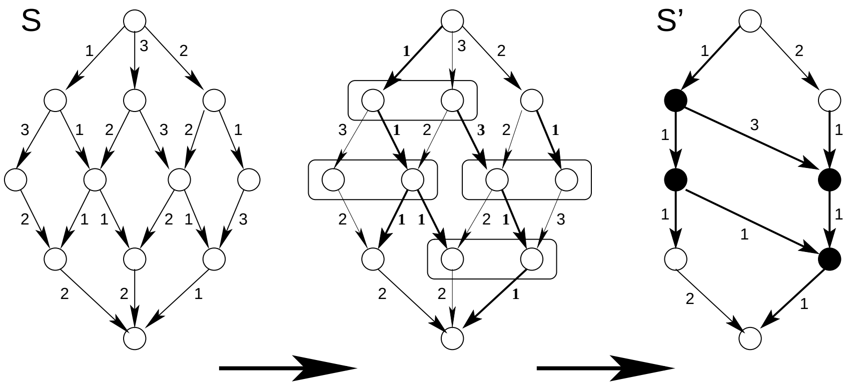

Fig. 1. State Space Lattice Abstraction

the initial and terminal vertices. A domain abstraction is a map from this set of labels to another set of labels of smaller cardinality. We use different domain abstractions at different levels. Let us consider two specific examples of domain abstractions for a 6 vertex SOP.

$$\begin{array}{r|cccccc} v & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline \phi_2(v) & 0 & 1 & 1 & 2 & 1 & 3 \\ \phi_3(v) & 0 & 1 & 2 & 2 & 3 & 4 \end{array} \tag{1}$$

ϕ_2 and ϕ_3 are applied to levels 2 and 3 respectively. Let $\times$ represent a non-existing edge in S and -1 correspond to a precedence constraint as we described earlier.

$$C'_{2,3} = \begin{array}{r|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \phi_3 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & 0 & 1 & 2 & 2 & 3 & 4 \\ \hline \phi_2 & & & & & & \\ 0 \to 0 & \times & 2 & \boxed{\mathbf{1}} & \boxed{\times} & 4 & \times \\ 1 \to 1 & \boxed{\mathbf{-1}} & \boxed{\times} & \boxed{\mathbf{1}} & \boxed{4} & \boxed{\mathbf{2}} & \boxed{\times} \\ 2 \to 1 & \boxed{-1} & \boxed{\mathbf{1}} & \boxed{\times} & \boxed{3} & \boxed{3} & \boxed{\mathbf{0}} \\ 4 \to 1 & \boxed{-1} & \boxed{2} & \boxed{1} & \boxed{3} & \boxed{\times} & \boxed{1} \\ 3 \to 2 & -1 & -1 & \boxed{\mathbf{3}} & \boxed{\times} & 2 & 3 \\ 5 \to 3 & -1 & -1 & \boxed{\mathbf{-1}} & \boxed{-1} & -1 & \times \end{array} \Longrightarrow \begin{pmatrix} \times & 2 & \mathbf{1} & 4 & \times \\ \mathbf{-1} & \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\ -1 & -1 & \mathbf{3} & 2 & 3 \\ -1 & -1 & \mathbf{-1} & -1 & \times \end{pmatrix} \tag{2}$$

Equation 2 shows how the original cost matrix is transformed into the cost matrix $C'_{2,3}$ that describes the connectivity and costs between S'_2 and S'_3. In order to guarantee lower bounds, the most conservative choice is applied to derive the cost representing the merged edges. If all edge costs are -1, then the resulting edge cost is also a -1 (precedences can be preserved). If some are -1 and some are $\times$ we must choose an $\times$ (and eliminate some precedences). If all preimage edges are $\times$ then so is the abstract edge cost (no new edge created). If there are some non -1 and non $\times$ entries then we must choose the one with the minimum cost. In [7], we prove that this construction results in valid lower bounds for S. Although conceptually S' is obtained by compressing S, S is so large that it cannot be enumerated and we build S' by expanding it from the bottom up while also calculating the lower bounds corresponding to the completion costs of the abstract partial tours of S'. For details see [7]. To obtain the abstractions, we minimize a user provided error function of the merged edges. In our experiments, we found that minimizing the absolute error between the merged edge costs and the new edge cost and applying some additional user provided penalty for eliminated precedences is particularly effective. We also used *transposition tables* at levels 3 and 4 to keep track of already visited states and to avoid their re-expansion.

4 Search

Our search algorithms are all derived from depth-first branch&bound. We branch depth-first because memory is reserved to store lower bounds. For the partial tour $s_l \in S_l$, let

$$f(s_l) = c(s_l) + b(s_l) \tag{3}$$

be an estimate of the total cost of the completed tour which is comprised from the cost of the partial tour $c(s_l)$ and its estimated completion cost $b(s_l)$. If $f(s_l)$ exceeds the cost of the best known feasible solution (the incumbent) then it can be pruned. The branches are also sorted in increasing f value order. Plain branch&bound can be greatly improved because of two special properties of the SOP. First, the SOP can be solved by searching from both directions and second, any consecutive sequence of vertices in any feasible tour also corresponds to a smaller SOP by itself. The cost matrix of the subproblem SOP can be obtained by readily taking entries from the original cost matrix and this subproblem can be solved independently. In our experiments we found that search in one direction is often much faster than in the other and therefore it is worthwhile searching from both directions. The switch from one direction to the other is triggered by reaching a user set node expansion limit. The search in both directions works on refining the same incumbent. We named this algorithm bi-directional depth-first branch&bound (or BDFBB). Recursive depth-first branch&bound (or RDFBB) uses a user imposed tolerance limit L. When the number of nodes expanded in the search under a subtree T exceeds L, the subproblem SOP corresponding to the largest subtree not including the start state but including T is solved independently with its own automatically generated lower bounds. We also employed a 3-opt on each new incumbent to opportunistically improve it (also used by [2,6,11]).

Max Width	Random			Absolute Error		
	b_0	ave(b)	Time	b_0	ave(b)	Time
500	400	402.90	0.57	53,965	53,704.10	0.57
2,000	490	830.25	0.80	**80,690**	59,256.60	0.77
8,000	545	1,219.24	1.00	54,115	52,356.10	1.13
32,000	630	2,146.16	1.46	**80,890**	65,315.20	1.29
128,000	680	2,934.33	2.13	**81,055**	60,713.50	2.24
512,000	710	4,925.33	3.30	**81,110**	69,782.70	2.63
2,048,000	965	7,841.39	5.82	**81,350**	65,885.20	4.48
8,192,000	1,265	9,704.65	18.43	**81,585**	70,473.30	12.87

Table 1. Generating Lower Bounds for `p43.4`

5 Results and Discussion

The two control parameters we supply to calculate the lower bounds are the maximum number of abstract states on a single level (maximum width) and the error function which the abstractions between consecutive levels optimize. As we have mentioned, minimizing absolute error (AE) proved to be effective. This is very clear from Table 1 which shows the average value of lower bounds (ave(b)), the lower bound corresponding to the start state (b_0) and the time it took in seconds to build the pattern databases of various sizes. The problem is one of the previously unsolved ones called `p43.4` from the TSPLIB [12]. b_0 is also a lower bound for the SOP itself. The bounds which are tighter than the previously best known value of 69,569 are shown in bold font. We compare randomly chosen abstractions to the ones that minimize AE. The values reported in the columns under "Random" correspond to the best from a pool of 20. Our smallest AE pattern database has much higher lower bounds than the best of 20 random ones which is also 16,384 larger. This experiment also reveals an interesting phenomenon that we encountered on more than one occasions. The AE pattern database with width 2000 is better than many of the larger ones minimizing the same error measure. At this time we have no explanation other than the abstractions seem to preserve the costs between the levels particularly well. With RDFBB, in 22 hours of CPU time we derived that the value of 83,005 is optimal[1]. We also have matched the best reported upper bounds within 60 minutes of CPU time to the open problems `p43.2`, `p43.3`, `p43.4`, `ry48p.2`, `ry48p.3`, `ry48p.4`, `ft53.2` and `ft53.3`. In these experiments we used BDFBB and also utilized a 3-opt edge exchange heuristic. We also generated a problem set of 16 SOPs; each over 30 vertices. These are derived from 4 base problems. Two correspond to rounded Euclidean distances on a 500×500 and a 6×6 grid with some random noise

[1] which contradicts a previously reported upper bound of 82,960 whose origin we could not trace

SOP		50,000		250,000		1,250,000	
		1,000 Nodes	Time (sec)	1,000 Nodes	Time (sec)	1,000 Nodes	Time (sec)
500×500	1	3,046,341	21,790.11	20,596	208.29	964	10.57
	2	212,656	1,132.43	8,761	52.63	692	5.27
	3	66,025	278.13	7,750	37.92	460	3.07
	4	27,796	127.25	9,288	49.80	488	3.17
6×6	1	9,022,057	66,451.02	1,775,135	16,058.50	141,987	1,570.72
	2	1,154,906	5,818.16	35,283	206.61	3,476	25.25
	3	26,579	133.14	9,509	57.88	814	5.63
	4	988	4.84	176	1.06	4	0.03
0-1000	1	360,391	2,773.21	34,060	336.05	17,091	184.29
	2	25,661	152.81	3,787	26.34	159	1.51
	3	266,526	1,243.66	70,435	376.39	8,936	60.89
	4	176	1.04	32	0.22	4	0.04
0-10	1	118,071	721.58	33,489	248.34	3,006	32.74
	2	468	3.06	202	1.48	54	0.52
	3	59,310	290.84	22,045	126.68	10,943	61.89
	4	7,788	40.01	895	5.33	43	0.32

Table 2. Nodes Expanded and CPU Times of the Problem Set

added to third of the edge costs. The other two are uniform random costs in $[0, 1000]$ and $[0, 10]$. Next, we generated four random precedence graphs. We applied the constraints implied by these to each of our four base problems and obtained SOPs with 107 (1), 160 (2), 210 (3) and 253 (4) precedence constraints. These include the 57 trivial precedences which are due to the fact that the start and terminal vertices are known. For ease of reference, we named these 16 test problems suggestively. First, we investigate solving these problems with three different size pattern databases of exponentially larger sizes. To compare them we measure nodes expanded in the search and CPU time. For these experiments we did not use the 3-opt tour improvement, only pattern databases. The results are tabulated in Table 2. Our results show that having larger databases pays off. In the case of 500×500-1 allocating a 5 times larger database results in a 148 fold reduction in nodes expanded. Increasing this size yet another 5 times and there is another 21-fold reduction in nodes expanded. While these very high ratios are not typical, our experiments indicate that in general the trade-off between memory and search speed favors adding more memory. It is also the case that, in most cases, the search finishes much faster when the SOP is more constrained. The branch&cut technique of Ascheuer et al. [2] solves the less constrained SOPs faster. The limiting factors in branch&cut, besides the dimensionality, are the size of the pool of inequalities that have to be searched and the construction of the facet inducing inequalities. Non of these are eased by more precedences. For us, more precedence constraints mean less branching in the search space

SOP		50,000					
		Plain		Rec		Rec + Trans	
		K Nodes	Time	K Nodes	Time	K Nodes	Time
	1	3,046,341	21,790.11	405,420	3,220.54	350,431	2,817.33
500×500	2	212,656	1,132.43	351,017	2,113.21	315,387	1,922.89
	3	66,025	278.13	33,327	146.79	26,871	118.18
	4	27,796	127.25	56,153	248.63	44,182	198.21
	1	9,022,057	66,451.02	18,950,825	134,032.34	12,250,761	101,120.71
6×6	2	1,154,906	5,818.16	30,4600	1,750.83	26,8862	1,551.40
	3	26,579	133.14	26,579	135.48	21,293	108.86
	4	988	4.84	988	4.96	988	4.98
	1	360,391	2,773.21	347,409	2,732.30	317,975	2,506.07
0-1000	2	25,661	152.81	17,865	112.49	17,686	109.62
	3	266,526	1,243.66	**51,707**	**230.21**	**48,021**	**212.11**
	4	176	1.04	176	1.03	176	0.03
	1	118,071	721.58	61,936	420.93	61,798	423.42
0-10	2	468	3.06	468	3.11	468	3.13
	3	59,310	290.84	**22,042**	**111.54**	**20,171**	**101.83**
	4	7,788	40.01	7,788	40.95	7,786	40.93

Table 3. Nodes Expanded and CPU Times of the Problem Set

and therefore less node expansions. Table 3 compares plain DFBB (Plain) to RDFBB (Rec) and RDFBB with transposition tables (Rec + Trans) with the smallest (maximum width 50,000) pattern databases. The values in bold show improvements that resulted in less search effort than searching with plain DFBB with a 5 times larger pattern database. The improvement is often remarkable but not always present. It requires further investigation to examine the cause as there is a large number of parameters to be set, such as the size of subproblems, databases and expansion limits.

With DFBB, the cost of the initial sequence of feasible solutions is very important since the costs are used as pruning bounds. Interestingly, we found that the size of the pattern database and even the magnitudes of the lower bounds stored have no relevance whatsoever when it comes to obtaining cheap solutions fast. Therefore the use of a 3-opt makes a very big difference. In fact, to this end we believe it would be more effective to start with the cheap and fast solutions of the HAS-SOP system [6] and instead of ordering by f values (Equation 3), we could make use of intermediate solutions of the LP relaxation of the branch&cut solver [2]. The solution to the LP relaxation is a fraction in $[0, 1]$ corresponding to each edge of the SOP. We could interpret these values as probabilities that tell how likely it is that the particular edge is included in the tour. We would sort branches by these probability values but still use our lower bounds for pruning.

We would also like to mention that our lower bounds could be generated when the precedences are arbitrary boolean constraints – such as "vertex

2 *or* 3 must precede vertex 4" and when some precedence violations are allowed when some user defined penalty is paid. In fact, we believe that pattern databases could be effective to generate lower bounds for a number of constrained Scheduling/Operations Research problems.

I would like to dedicate this work in the memory of my dear friend, Dr. Balázs Zombori, who recently died in a car accident.

This research was supported in part by a research grant provided by the Natural Sciences and Engineering Research Council of Canada. I would also like to thank *Dr. Robert C. Holte* (University of Alberta) for his insights and encouragement.

References

1. N. Ascheuer. *Hamiltonian path problems in the on-line optimization and scheduling of flexible manufacturing systems.* PhD thesis, Technical University of Berlin, 1995.
2. N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17(1):61–84, 2000.
3. N. Christofides, A. Mingozzi, and P. Toth. State space relaxation procedures for the computation of bounds to routing problems. In *Networks*, volume 11, pages 145–164, 1981.
4. J. C. Culberson and J. Schaeffer. Searching with pattern databases. In *Proceedings of the Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, volume 1081 of *LNCS*, pages 402–416, 1996.
5. L. F. Escudero, M. Guignard, and K. Malik. A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence constraints. In *Annals of Operations Research*, volume 50, pages 219–237, 1994.
6. L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.
7. I. T. Hernádvölgyi. Solving the sequential ordering problem with automatically generated lower bounds. Technical Report TR03-16, University of Alberta, 2003.
8. R. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of the Workshop on Computer Games (W31) at IJCAI-97*, pages 21–26, 1997.
9. A. Mingozzi, L. Bianco, and S. Ricciardelli. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, 45:365–377, 1997.
10. W. Pulleyblank and M. Timlin. Precedence constrained routing and helicopter scheduling: Heuristic design. Technical Report RC17154, IBM, 1991.
11. Dong-Il Seo and Byung-Ro Moon. A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem. In *GECCO-2003*, volume 2723 of *LNCS*, pages 669–680. Springer-Verlag, 2003.
12. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/