Metaheuristics for the Asymmetric Hamiltonian Path Problem

João Pedro PEDROSO

INESC - Porto and
DCC - Faculdade de Ciências, Universidade do Porto, Portugal
`jpp@fc.up.pt`

Abstract. One of the most important applications of the Asymmetric Hamiltonian Path Problem is in scheduling. In this paper we describe a variant of this problem, and develop both a mathematical programming formulation and simple metaheuristics for solving it. The formulation is based on a transformation of the input data, in such a way that a standard mathematical programming model for the Asymmetric Travelling Salesman Problem can be used on this slightly different problem. Two standard metaheuristics for the asymmetric travelling salesman are proposed and analysed on this variant: repeated random construction followed by local search with the 3-Exchange neighbourhood, and iterated local search based on the same neighbourhood and on a 4-Exchange perturbation. The computational results obtained show the interest and the complementary merits of using a mixed-integer programming solver and an approximative method for the solution of this problem.

1 Introduction

We are dealing with the following problem: given an operation currently being done in a machine, determine the order for the set of operations to be produced next, such that the total production time is minimized. There are no precedence constraints among the operations, but there are changeover times which depend on the production sequence. Minimizing the total production time is equivalent to minimizing the time spent in changeovers, as the other times are constant.

This problem is relevant in many practical situations. In paint production the machine cleaning times are usually dependent on the sequence; for example, producing white colour after grey requires a much more careful cleaning than the other way around. The production of steel is also a situation where the sequence of production is very important, having very strict rules and costs that depend on the order. Yet another practical application is in food manufacturing, where strong flavours can be produced after flavourless products at a small cost, but very careful and lengthy cleaning is required in the inverse situation.

One possibility for modelling this problem is to consider a graph with a node for each of the items that must be produced. There are two arcs between every pair of nodes, one in each direction, representing the changeover time between the corresponding products. A solution to the original problem corresponds to determining a Hamiltonian path in this graph, i.e., a path going through all the nodes in the graph. The path must start with a particular node (the item being currently produced), but there is no concern about the ending node. Let us call this the *"Fixed-Start Asymmetric Hamiltonian Path" (FSAHP)* problem.

Given the similarity of this problem with the Travelling Salesman Problem, in particular with its asymmetric variants, we considered adapting the methods that have been developed for that problem to the current situation. Throughout this paper we will describe more formally the problem in mathematical programming, explain in detail the metaheuristics that we implemented for solving it, and present the results of applying it to a set of benchmark problems.

2 Problem description

We are given a graph $G(V, A)$ where V is the set of nodes and A the set of arcs. In the classical Asymmetric Travelling Salesman Problem (ATSP), nodes correspond to cities to be visited and arcs to the distance between them. In our case, each node represents a product to be manufactured, every arc (i, j) has a cost $D(i, j)$ corresponding to the (asymmetric) changeover time between product i and j, and there is a special node v_1, which must be the first node in the path, and corresponds to the last previously manufactured product (or to the city where the salesperson currently is, the classical problem).

With simple data preprocessing, standard ATSP formulations can be adapted to the current problem, as shown below.

Property 1. Redefine the distance from any node to the first (fixed) node in the path, v_1, as zero (all other distances remaining unchanged). A minimum Hamiltonian cycle determined with this data defines a path which is an optimal solution to the FSAHP, with the same optimal objective value.

Proof. Let us call the optimal solution to the FSAHP $(p_1, \ldots, p_n)$; this is a path, with $p_1 = v_1$, covering all the nodes. This path can be extended into a cycle, without increasing the cost, by adding the arc (p_n, v_1).

Suppose there is a cycle $(s_1, \ldots, s_n, s_1)$, with $s_1 = v_1$, with a smaller objective; then, as the arc (s_n, v_1) has zero length, the path $(s_1, \ldots, s_n)$ would have to be shorter than $(p_1, \ldots, p_n)$. But in this case $(s_1, \ldots, s_n)$ would be a better solution to the FSAHP than $(p_1, \ldots, p_n)$, contradicting the assumption.

2.1 Formulation in mathematical programming

There are many formulations for the ATSP, and their study is an active field in mathematical programming. For the purposes of this paper, we will restrict to the most common one, due to [1]:

$$\text{minimise} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n$$
$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n$$
$$(n-1)x_{ij} + u_i - u_j \leq n-2, \quad i,j = 2, \ldots, n$$
$$x_{ij} \in \{0,1\}, \quad i,j = 1, \ldots, n, \quad u_i \in \mathbb{R} \quad i = 1, \ldots, n$$

The optimal cycle is the set of arcs (i,j) such that $x_{ij} = 1$. The solution to the FSAHP is the n-node path starting with v_1 in this cycle.

3 Basic heuristics and local search

The most straightforward way for solving the Fixed-Start Asymmetric Hamiltonian Path with heuristics and metaheuristics is to apply the transformation on the data proposed in Section 2, and solve an Asymmetric Travelling Salesman. Then, the solution to the original problem is obtained by selecting the n-node path starting with v_1 in the ATSP's solution.

The characteristics of the path problem could be exploited for devising more adapted neighbourhoods, but it turns out that the performance degrades in most of the studied instances, possibly due to the losing symmetry properties.

3.1 Construction

Simple construction heuristics for the ATSP are based on equivalent heuristics for the symmetric TSP (nicely described e.g. in [2]). As for the metaheuristics described in this paper, the initial solution is constructed based on a random permutation of $\{1, \ldots, n\}$.

3.2 Improvement

The most common improvement methods for problems related to the TSP are based on *exchange heuristics*: remove k edges, breaking the cycle tour into k paths; then reconnect those paths into a different cycle [3, 4]. For the symmetric TSP, the most commonly used neighbourhood is 2-Exchange: remove two non-consecutive edges, and add two other edges, as shown in Figure 1.

As for the ATSP, there are no 2-Exchange moves that keep path orientation, and hence they are not usually employed [5]. The most commonly used moves are 3-Exchange, keeping path orientation, as shown in Figure 2.

For implementing local search based in this neighbourhood in an efficient way, moves that are known not to lead to improvements should be avoided. For this purpose, the list of the neighbours of a given vertex, sorted by distance, is searched only up to a certain point.

Let us first recall what is commonly done with the (symmetric) TSP. Consider a tour represented by $p = (p_1, p_2, \ldots, p_n)$, and let us denote the last element of p as either p_n or p_0. Each edge (p_{i-1}, p_i), for $i = 1, \ldots, n$, is examined for improving exchanges, through removing it and another edge (p_{j-1}, p_j), and adding two different edges, in such a way that a new tour is formed (p_j must be separated from p_i by at least two nodes). A new tour is constructed by adding edges $\{p_{i-1}, p_{j-1}\}$ and $\{p_i, p_j\}$.

Property 2. For a given i, improving moves cannot be missed if j is restricted to:

1. nodes connected to p_{i-1} such that their distance to p_{j-1} is smaller than $D(p_{i-1}, p_i)$;
2. nodes connected to p_i such that their distance to p_i is smaller than $D(p_{i-1}, p_i)$.

Proof. Let $p_{i-1}, p_i, p_{j-1}, p_j$ be represented by a, b, c, d, respectively, as in Figure 1. In an improving move there must be $D(a,c) + D(b,d) < D(a,b) + D(c,d)$, implying that either $D(a,d) < D(a,b)$ or $D(c,b) < D(c,d)$, or both. Hence, in an improvement, at least one of the added edges must be smaller than at least one of the removed edges. The case of an added edge being smaller than $\{a,b\}$ is examined by considering all edges $\{a,c\}$ such that $D(a,c) < D(a,b)$, and all edges $\{b,d\}$ such that $D(b,d) < D(a,b)$. The remaining potential improvement case corresponds to having the edge $\{c,d\}$ larger than either $\{a,c\}$ or $\{b,d\}$; but this possibility is examined for i such that $c = p_{i-1}$ and $d = p_i$.

Let us now go back to the ATSP problem and the 3-Exchange neighbourhood. Consider a tour represented by $p = (p_1, p_2, \ldots, p_n)$. Each arc (p_{i-1}, p_i), for $i = 1, \ldots, n$, is examined for improving exchanges, through removing it and other two arcs (p_{j-1}, p_j) and (p_{k-1}, p_k). A new tour is constructed by adding arcs (p_{i-1}, p_j), (p_{j-1}, p_k), and (p_{k-1}, p_i).

Property 3. For a given i, improving moves cannot be missed if j and k are restricted as follows:

1. j is restricted to nodes outgoing from p_{i-1} such that their distance from p_{i-1} is smaller than $D(p_{i-1}, p_i)$; furthermore, in this case k is restricted to nodes outgoing from p_{j-1} such that distance $D(p_{i-1}, p_j) + D(p_{j-1}, p_k)$ is smaller than $D(p_{i-1}, p_i) + D(p_{j-1}, p_j)$, and p_k is not in the path from p_i to p_{j-1}.
2. $k - 1$ is restricted to nodes incoming into p_i such that their distance to p_i is smaller than $D(p_{i-1}, p_i)$; furthermore, in this case j is restricted to nodes incoming into p_k such that distance $D(p_{k-1}, p_i) + D(p_{j-1}, p_k)$ is smaller than $D(p_{i-1}, p_i) + D(p_{k-1}, p_k)$, and p_j is not in the path from p_k to p_{i-1}.

Proof. Let $p_{i-1}, p_i, p_{j-1}, p_j, p_{k-1}, p_k$ be represented by a, b, c, d, e, f, respectively, as in Figure 2. In an improving move there must be $D(a,d) + D(c,f) + D(e,b) < D(a,b) + D(c,d) + D(e,f)$, implying that at least one of the added arcs must be smaller than at least one of the removed ones.

Let us consider an improving move for which either $D(a,d) + D(c,f) > D(a,b) + D(c,d)$, or $D(a,d) > D(a,b)$. In the former case, there must be $D(e,b) < D(e,f)$, and

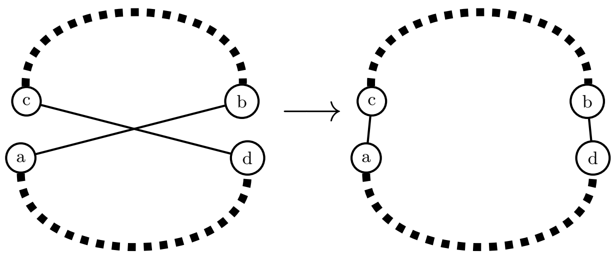

Fig. 1. Single 2-Exchange possibility for the (symmetric) TSP. Edges $\{a, b\}$, $\{c, d\}$ are removed, and replaced by $\{a, c\}$, $\{b, d\}$.

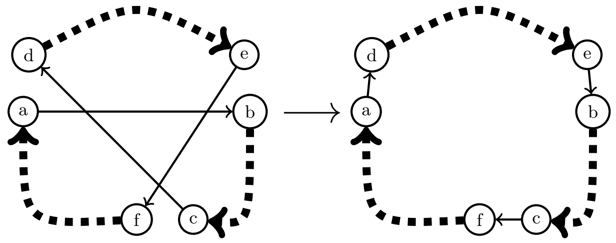

Fig. 2. Single 3-Exchange possibility without path inversions for the ATSP. Arcs (a, b), (c, d), (e, f) are replaced by (a, d), (e, b), (c, f).

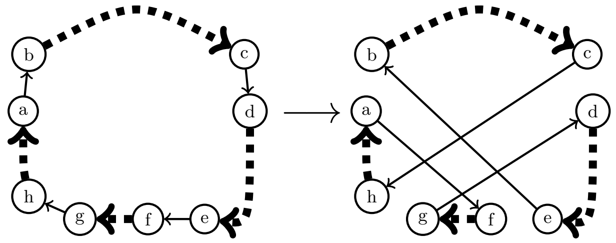

Fig. 3. A 4-Exchange (double bridge) movement without path inversions for the ATSP, as implemented in iterated local search.

this is tackled in the main i cycle, for $i : p_{i-1} = e$. As for the latter case, there must be $D(c, f)+D(e, b) < D(c, d)+D(e, f)$; thus, either $D(c, f) < D(c, d)$, or $D(e, b) < D(e, f)$, or both. But this situation is tackled for $i : p_{i-1} = c$ or $i : p_{i-1} = e$, respectively.

3.3 Implementation

In our implementation, indices for the outer cycle (i) are searched in random order. Indices j and k are search by increasing distance to nodes p_{i-1} and p_i, until reaching the limits defined by Property 3. Improvements are accepted in an *first-improve* manner, i.e., an improving movement is immediately accepted. The initial solution is a random permutation of $\{1, \ldots, n\}$.

3.4 Improved heuristics

Random-start local search: in this metaheuristics, the following steps are repeated until reaching a stopping criterion (in our implementation, exceeding the limit CPU time):

1. create a random solution;
2. improve it until reaching a local optimum;
3. possibly, update the best solution found so far.

Iterated local search: for this metaheuristics, after reaching a local optimum a deep modification on the solution structure is introduced; the solution thus obtained is then improved until reaching another local optimum, and the whole process is repeated until reaching the stopping criterion.

The deep modification made at each iteration is a 4-Exchange movement, as depicted in Figure 3. This is usually called a "*double bridge*" movement. Our implementation of iterated local search consists of obtaining a random starting solution, and then repeating the following steps:

1. improve the solution until reaching a local optimum;
2. possibly, update the best solution found so far;
3. randomly select 4 arcs in the solution; exchange them with 4 different arcs, in such a way that a tour (with no path inversions) is formed.

4 Results

The metaheuristics proposed in this paper were compared to a mixed-integer programming (MIP) solver, through an experiment with a set of standard benchmarks instances. These correspond a modification of the ATSP instances available in the TSPLIB [6]; the starting node v_1 is the first city in the instance, and, for tackling the path problem, the distances from any other node to v_1 are redefined as zero (as described in Section 2). The experiment was run in a computer with a Quad-Core Intel Xeon, 2.66 GHz processor, running the Mac OS X operating system version 10.6.3; only one CPU was allocated to this experiment. The MIP solver used is GUROBI [7], one of the leading commercial solvers. Metaheuristics were implemented in the Python programming language, version 2.6.1; this is considerably slower than the compiled, executable code of GUROBI. Hence, results are not truly comparable; however, they still allow drawing many interesting conclusions. In all the experiments, the CPU time for an observation of a method solving an instance was limited to about 300 seconds; as for the metaheuristics, the results correspond to the minimum, average, and maximum of 10 independent observations. The results are presented in table 1.

The first interesting conclusion is that a state-of-the-art MIP solver can reach the optimum for many of the benchmark instances (those for which the lower bound obtained is identical to the upper bound); this is an enormous progress with respect to some years ago. In these cases, both metaheuristics could also find systematically the optimum, except for instances of the `ftv` series. For these instances and `atex5`, the result of the MIP solver is better than the average solution of each metaheuristics; for all the other instances, both metaheuristics are better.

A very interesting result was obtained for instances `rbg403` and `rbg443`; indeed, even though no feasible solution was found by the MIP solver in the

	Multi-start local search			Iterated local search			GUROBI	
Instance	minimum	average	maximum	minimum	average	maximum	LB	UB
atex1	1564	1564	1564	1564	1564	1564	1564	1564
atex3	2342	2342	2342	2342	2342	2342	2342	2342
atex4	2681	2681	2681	2681	2681	2681	2681	2681
atex5	4659	4663.8	4669	4659	4670.8	4747	4595	4659
atex8	41531	41763	41960	41299	41598.8	41900	1027	∞
big702	78933	79081.4	79316	78492	78847.4	79518	$-\infty$	∞
br17	27	27	27	27	27	27	27	27
code198	4541	4541	4541	4541	4541	4541	4541	4541
code253	106957	106957	106957	106957	107032	107333	105716	∞
dc112	10916	10919.3	10922	10914	10916.7	10919	10860	10968
dc126	120725	120770	120827	120709	120754	120808	119702	126506
dc134	5543	5544.6	5547	5539	5540.8	5542	5529	∞
dc176	8402	8406.3	8410	8400	8403.3	8409	8356	∞
dc188	9977	9979.9	9986	9974	9979.8	9988	9911	∞
dc563*	25880	25880	25880	25880	25880	25880	25687	∞
dc849	37496	37501.7	37506	37488	37498.6	37504	$-\infty$	∞
dc895*	106963	106963	106963	106963	106963	106963	$-\infty$	∞
dc932*	478316	478316	478316	478316	478316	478316	$-\infty$	∞
ft53	6099	6099	6099	6099	6099	6099	6099	6099
ft70	37230	37231.2	37234	37230	37230.4	37234	37228	37230
ftv100	1743	1746.5	1747	1743	1744.7	1747	1743	1743
ftv110	1908	1910.6	1914	1908	1912.3	1917	1900	1908
ftv120	2074	2078.2	2081	2074	2074.5	2077	2074	2074
ftv130	2240	2250.2	2262	2240	2242.7	2250	2240	2240
ftv140	2358	2364.4	2375	2358	2360.1	2366	2356	2356
ftv150	2547	2554.5	2563	2547	2548.1	2550	2547	2547
ftv160	2600	2605.5	2616	2600	2603.1	2605	2600	2600
ftv170	2690	2701.7	2717	2689	2691.4	2694	2668	2713
ftv33	1223	1223	1223	1223	1223	1223	1223	1223
ftv35	1363	1363	1363	1363	1363	1363	1363	1363
ftv38	1438	1438	1438	1438	1438	1438	1438	1438
ftv44	1535	1535	1535	1535	1535	1535	1535	1535
ftv47	1689	1689	1689	1689	1689	1689	1689	1689
ftv55	1539	1539	1539	1539	1539	1539	1539	1539
ftv64	1726	1726	1726	1726	1726	1726	1726	1726
ftv70	1881	1881	1881	1881	1881	1881	1881	1881
ftv90	1538	1538	1538	1538	1538	1538	1538	1538
kro124p	35584	35584	35584	35584	35584	35584	35581	35584
p43	589	589	589	589	589	589	549	589
rbg323	1308	1308	1308	1308	1308	1308	1308	1308
rbg358	1143	1143	1143	1143	1143	1143	1143	1143
rbg403	2450	2450	2450	2450	2450	2450	2450	∞
rbg443	2710	2710	2710	2710	2711.7	2719	2710	∞
ry48p	13870	13870	13870	13870	13870	13870	13869	13870
td100.1	267047	267047	267047	267047	267047	267047	267047	267058
td1000.20	1241220	1241230	1241230	1241220	1241230	1241230	$-\infty$	∞
td316.10	688929	688929	688929	688929	688929	688929	688929	688929

Table 1. Results obtained using multi-start local search, iterated Local search, and the lower and upper bounds obtained by the MIP solver *GUROBI*, for a CPU limit of 300 seconds. (Instances `dc563, dc895, dc932` were allowed only one descent, as it takes more than 300 seconds.)

CPU time allowed, the best solution found by metaheuristics can be proven optimal, as its objective value equals the MIP lower bound.

As for the comparison between the two metaheuristics proposed, iterated local search is at least as good as multi-start local search for most instances, being strictly better for many of them; the slight increase in complexity seems, hence, to be worthy.

5 Conclusions

In this paper we describe a variant of the Asymmetric Hamiltonian Path Problem, with applications in scheduling. We present a mathematical programming formulation, and simple approximative methods for solving it. The metaheuristics are random-start local search and iterated local search; both of them provided very good results, with a slight advantage to the latter.

For easy problems a mixed-integer programming solver could find the optimum in a relatively small time; for larger, more difficult problems, the approximative methods could find better solutions in the CPU time allowed.

Improvements on the metaheuristics are expected if “don’t look bits” are used, in order to keep track of cities for which search could be skipped. Another possible improvement concerns limiting the number of neighbours of each city that are allowed to be explored for exchanges. Both of these modifications may provide a considerable speedup, at the cost of, possibly, loosing local optimality.

Acknowledgments. This research was supported in part by FCT – Fundação para a Ciência e a Tecnologia (Project **PTDC/GES/73801/2006) and by the European project CIVITAS-ELAN, under Framework Programme 7.

Our special thanks to Prof. Nelma Moreira for proof reading this manuscript.

References

1. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. J. ACM **7**(4) (1960) 326–329
2. Johnson, D., McGeoch, L.: Local search in combinatorial optimization. In Aarts, E., Lenstra, J.K., eds.: Local search in combinatorial optimization. John Wiley & Sons, Inc., New York, NY, USA (1997)
3. Croes, G.A.: A method for solving traveling-salesman problems. Operations Research **6** (1958) 791–812
4. Flood, M.M.: The traveling-salesman problem. Operations Research **4** (1956) 61–75
5. Johnson, D.S., Gutin, G., McGeoch, L.A., Yeo, A., Zhang, W., Zverovitch, A.: Experimental analysis of heuristics for the atsp. In Gutin, G., Punnen, A.P., eds.: The Traveling Salesman Problem and Its Variations. Volume 12 of Combinatorial Optimization. Kluwer Academic Publishers, Boston, USA (2002)
6. Bixby, B., Reinelt, G.: TSPLIB – A library of travelling salesman and related problem instances. Internet repository (1995) http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.
7. Gurobi Optimization, Inc.: Gurobi Optimizer Reference Manual, Version 2.0, http://www.gurobi.com. (2010)