Graph Transformations for the Vehicle Routing and Job Shop Scheduling Problems*

J. Christopher Beck[1], Patrick Prosser[2], and Evgeny Selensky[2]

[1] ILOG SA, Paris, France. cbeck@ilog.fr
[2] Department of Computing Science, University of Glasgow, Scotland.
pat/evgeny@dcs.gla.ac.uk

Abstract. The vehicle routing problem (VRP) and job shop scheduling problem (JSP) are two common combinatorial problems that can be naturally represented as graphs. A core component of solving each problem can be modeled as finding a minimum cost Hamiltonian path in a complete weighted graph. The graphs extracted from VRPs and JSPs have different characteristics however, notably in the ratio of edge weight to node weight. Our long term research question is to determine the extent to which such graph characteristics impact the performance of algorithms commonly applied to VRPs and JSPs. As a preliminary step, in this paper we investigate five transformations for complete weighted graphs that preserve the cost of Hamiltonian paths. These transformations are based on increasing node weights while reducing edge weights or the inverse. We demonstrate how the transformations affect the ratio of edge to node weight and how they change the relative weights of edges at a node. Finally, we conjecture how the different transformations will impact the performance of existing VRP and JSP solving techniques.

1 Introduction

The vehicle routing problem (VRP) and the job shop scheduling problem (JSP) are two common combinatorial problems that have natural graphical representations. In fact, these two problems have a number of similarities both in their graphical representations and on a conceptual level. While the structure of the graphs is largely similar (and is, indeed, identical for core subproblems), the graph characteristics tend to be different. In particular, the difference that forms the focus of the work in this paper is that graphs representing VRP problems tend to have very high edge weights relative to node weights while JSP graphs are the reverse.

It is our conjecture that we can develop a deeper understanding of the models and algorithms for VRPs and JSPs by examining and transforming the characteristics of the underlying graph representations and by experimenting with the transformed problems. In this paper, we present five graph transformations each of which is based on increasing node weights while decreasing edge weights or

* This work was supported by EPSRC research grant GR/M90641 and ILOG SA.

the inverse. From the perspective of VRPs and JSPs, the transformations that increase node weights relative to edge weights will make the graph look more like a JSP problem than a VRP. Similarly, the inverse transformation will give the problem characteristics that are more typical of VRPs. Our long term research direction, therefore, is to identify interesting problem reformulations (perhaps based on graph transformations), apply them to VRP and JSP problems, and empirically analyze the performance of existing VRP and JSP techniques.

Previous work provides a basis for our intuitions and forms a preliminary analysis. In Beck et al. [2] the three order-dependent transformations presented below were applied to VRP problems. It was shown that:

- a guided local search based VRP solving technique significantly outperforms a constructive, constraint programming based scheduling technique when applied to VRP problems;
- the VRP solving technique sometimes performs worse when the problems are transformed by the order-dependent transformations than it performs on the non-transformed problems;
- the JSP solving technique performs marginally better on the transformed problems than on the non-transformed problems.

While this work is interesting, no systematic analysis of the changes in the graph characteristics induced by the transformation techniques was performed. In addition, neither the direct order independent nor inverse transformations presented below were tested. In this paper, we present the graph transformations in a more general framework and perform a systematic analysis of their impact on complete weighted graphs. Each of these transformations preserves the cost of Hamiltonian paths. We are interested in the extent to which the transformations change the ratio of edge weights to node weights and in the extent to which the relative weights of edges are changed. It is our intuition, based on preliminary empirical evidence, that these two characteristics have an impact on VRP and JSP solving techniques.

In the following section, we define VRPs and JSPs in more depth together with their standard graphical representations. We also discuss some similarities and differences between these problems which provide motivation for our interest in graph transformations. We then present the graph transformations addressed in this paper followed by an empirical study of the impact of the transformations on characteristics of the graphs. We conclude by making some conjectures regarding how the use of the transformations may affect the performance of existing VRP and JSP solving techniques.

2 Vehicle Routing and Job Shop Scheduling

In the *delivery* variant of the *vehicle routing problem* (VRP), m identical vehicles initially located at the depot are to deliver discrete quantities of goods to n customers. The locations of the depot and customers are known. Each customer has a discrete demand for goods and each vehicle has a discrete capacity, i.e., it can carry quantities less than or equal to its capacity. A vehicle can make only one tour starting at the depot, visiting a subset of customers and returning to the depot. It is also assumed that travel time equals travel distance computed using the Euclidean metric. A solution consists of a set of tours for a subset of vehicles such that (i) all customers are served only once, (ii) the total distance traveled by the fleet is minimal. The problem is NP-hard [7].

An $n \times m$ job shop scheduling problem (JSP) consists of n jobs and m resources. Each job consists of a set of m completely ordered activities, where each activity has a duration for which it must execute and a resource which it must execute on. The complete ordering defines a set of precedence constraints, meaning that no activity can begin execution until the activity that immediately precedes it in the complete ordering has finished execution. Each of the m activities in a single job requires exclusive use of one of the m resources defined in the problem. No activities that require the same resource can overlap in their execution and once an activity is started it must be executed for its entire duration (i.e., no pre-emption is allowed). The job shop scheduling decision problem is to decide if all activities can be scheduled, given for each job a release date of 0 and a due date of the desired makespan, D, while respecting the resource and precedence constraints. The job shop scheduling decision problem is NP-complete [7].

While not part of the basic JSP definition, transition times and resource alternatives have been studied in the scheduling community [6]. In the case of a transition time, there is an additional temporal constraint specifying a minimum time interval that must expire between any pair of activities executed on the same resource. The time interval may differ for different pairs of activities. When alternative resources are represented, rather than having a single resource to execute on, each activity has a set of resources from which the one to execute on must be chosen.

2.1 Graphical Representations

In a graphical representation of a VRP, each node corresponds to a visit. The duration of the visit is the weight of the node. Edges represent the travel between visits, with the weight of the edges representing the time required for the travel. A solution is a set of cycles each of which begins and ends at the distinguished depot node. Aside from the depot node, each node appears in exactly one cycle. An optimal solution is one in which the total weight of the cycles (node weights plus edge weights) is minimized.

A JSP problem can also be represented as a graph. Each activity is represented by a node with the duration of the activity being the node weight. There are two types of edges:

- directed, conjunctive edges represent a precedence constraint specifying that one activity must precede another. Transition time is represented as the weight of the edge.
- undirected, disjunctive edges represent the fact that the activity pair must be ordered but that the specific ordering is not predefined in the problem definition.

A solution is the transformation of each disjunctive edge into a conjunctive edge such that the longest path in the graph is less than or equal to D.

We observe that the core problem of finding a path or cycle in a graph exists in both VRP and JSP. In a VRP we must find a set of cycles while in a JSP we need to find a set of paths, one for each resource, by transforming disjunctive edges into conjunctive edges. In this paper, we will concentrate on the subproblem of finding a single minimum cost Hamiltonian cycle in a complete weighted graph. This is equivalent to a VRP problem with a single vehicle and to a JSP problem with a single resource.

2.2 Motivation: VRP and JSP Similarities

As well as similar graph representations, on a conceptual level, vehicle routing problems and scheduling problems are similar. Consider the following:

- They both involve the execution of tasks (activities in the factory and visits in the VRP).
- A task can only be completed through the use of one or more resources (tools and machines on the shop floor, drivers and vehicles in the VRP).
- Resources are often constrained by capacity that specifies, for example, the number of tasks that can be processed.
- Often there is a set of alternative resources to choose from (similar machines on the shop floor, a fleet of vehicles in the VRP).
- When a task is completed there may be some interval of time that must expire before the resource can be used for another task, and this interval may depend on the pair of consecutive tasks (a transition time or set up cost on a machine in the factory, a travel distance between visits in the VRP).
- There may be further temporal constraints among the tasks, specifying time windows when they can and cannot be executed and/or specifying a necessary relationship between tasks (e.g., task B must not be executed until after task A is finished).
- The problem is solved when the resources have been assigned to tasks and the tasks have been assigned start times or ordered such that all temporal and capacity constraints are respected.
- There is an optimization criterion involving the minimization of various definitions of the length of time necessary to complete all the activities. For example, one may seek to minimize the sum of all transition times.

With these similarities, one might expect that similar technology is used to solve vehicle routing and scheduling problems. But this isn't the case. For example, local search techniques (tabu search, simulated annealing, guided local search) [8, 10, 16] are more popular for vehicle routing problems whereas complete and quasi-complete search techniques [9] are used more in scheduling. As a single data point, ILOG markets two commercial C++ libraries (ILOG Scheduler and Dispatcher) for solving scheduling and vehicle routing problems respectively. Though both products are built on constraint programming technology, the core technology in the scheduling product is global constraint propagation [11, 12] while the core technology in the vehicle routing product is local search [5].

On the other hand, one can see the following important differences between the two problems:

- Transition time *vs* activity duration - In scheduling problems the duration of an activity is typically much larger than the transition time. This is not the case in the VRP, where, in fact, travel is many times the visit duration. Furthermore, much of the research in scheduling has focused on techniques that reason directly about the durations of activities while putting less emphasis on the transition time. The state of the art in constraint-based scheduling is based to a large extent on global constraint propagation [12, 13] and heuristic search techniques [14, 1] that do not directly take into account transition time. Therefore, we expect that scheduling problems where the transition time appears to be a key aspect of finding a solution, will be difficult for current scheduling technology.
- Alternative resources - VRPs have many more alternative resources than are typically studied in scheduling problems. Although the vehicles might not all be used, a common benchmark set [15] starts with 20 or more vehicles. In contrast, existing scheduling research on resource alternatives typically has many fewer alternatives. For example, Focacci et al. [6] experimented on problems with up to three alternatives while Davenport & Beck [4] used problem instances of up to eight alternatives.
- Complex temporal relationships - Scheduling problems tend to have more complex temporal relationships among activities. As we have seen, precedence constraints among the activities in a job are part of the JSP. More complex metric temporal constraints have been widely explored [3] and strong global constraint propagation techniques that take such relationships into account exist [11]. The standard VRP has no temporal constraints.

What are the characteristics of vehicle routing problems that make them more amenable to local search techniques as opposed to construction methods? What properties of scheduling problems make them more suitable to systematic search and powerful constraint propagation?

In this paper, we focus on the issue of transition time *vs* duration. As noted above, previous work [2] supports the intuition that VRP and JSP techniques perform differently on VRP problems that have been transformed to increase node weight than they do on the original VRP problems. As a first step toward a systematic understanding of the reasons for these performance differences, in this paper we investigate how the transformations affect the graph characteristics.

3 Cost-preserving transformations of complete undirected graphs

We are interested in transformations that preserve the cost of any solution on the graph. In the following sections, for simplicity, a solution is assumed to be a cycle on the graph (i.e., we consider traveling salesman problem graphs; when presenting a transformation we then show how it carries over to the case of Hamiltonian paths). The cost of a solution is assumed to be the sum of the weights of the nodes and edges in the respective cycle.

First, we present transformations that reduce the edge weights on a graph by adding a portion of them to the node weights. We refer to these transformations as *direct*. There also exists an *inverse* transformation which similarly reduces the node weights and increases the edge weights.

3.1 Direct transformations

Order-dependent transformation Consider a traveling salesman problem where nodes have weights as well as edges[1]. In a solution we must visit each node once and once only. The cost of visiting a node is then the weight of the edge entering the node plus the weight of the edge exiting the node, plus the weight of the node itself. We can transform this cost such that the node weight is increased and the weights on the entering and exiting edges are reduced.

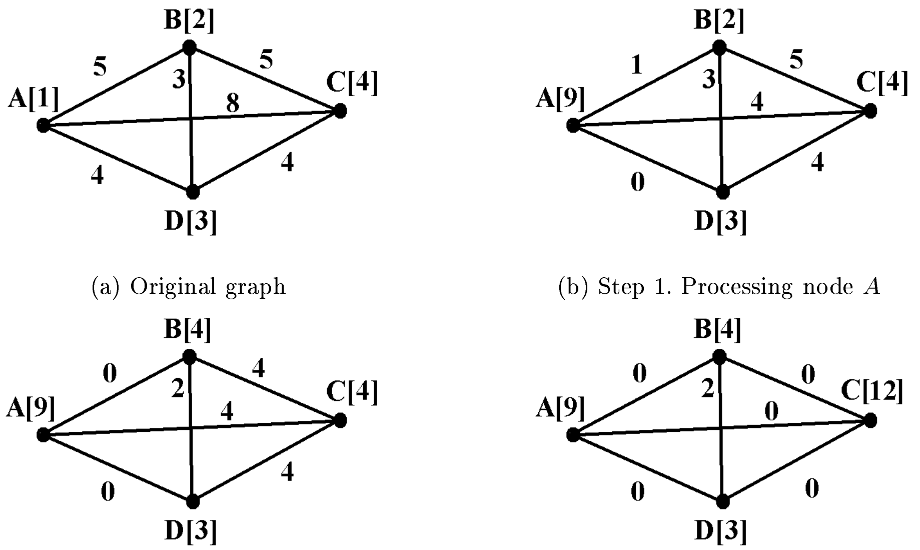
Consider node j, with entering edge (i, j) and exiting edge (j, k), with weights w_j, w_{ij} and w_{jk} respectively. Let $w_{min} = min(w_{ij}, w_{jk})$. We can reduce the weight of both edges by w_{min}, such that at least one of these becomes zero, and add to the node weight $2w_{min}$. This preserves the cost of entering, visiting, and exiting j. More generally, for a TSP we can process each node i as follows:

1. let w_{min} be the weight of the cheapest edge incident on node i;
2. for each edge incident on node i subtract w_{min} from the edge's weight;
3. add $\delta_i = 2w_{min}$ to w_i, the weight of node i;
4. the node now has at least one incident edge of zero weight.

We need only process each node once, i.e., after processing, a node has at least one zero-weight incident edge and re-processing will have no effect. Fig. 1 shows the sequence of transformations of a four-node clique. All nodes start with weights shown in square brackets and are processed in alphabetic order. We start with the initial problem and then process node A. This removes a weight of 4 from A's incident edges and adds a weight of 8 to node A (Fig. 1b). We then process node B. This reduces the weight of incident edges by 1 and adds a weight of 2 to node B (Fig. 1c). Then we move on to node C and reduce the weight of incident edges by 4 and add a weight of 8 to node C (Fig. 1d). Finally, processing D has no effect because there is a zero-weight incident edge. Note, that if we processed the nodes in a different order we might end up with a different final graph, i.e., the transformation is order-dependent.

In a Hamiltonian path we have two distinguished nodes, i.e., the start node s and end node e, and the transformation is then modified as follows. Since s is not entered and e is not exited, we do not add to the weights of those nodes twice the weight of their minimum incident edges. Instead, we add the minimum weight once only, and process all other nodes as above.

[1] For a conventional TSP node weights would then be zero.



(a) Original graph

(b) Step 1. Processing node A

(c) Step 2. Processing node B

(d) Step 3. Processing node C

Fig. 1. Transformation of a 4-node clique. Node weights are shown in square brackets.

As noted above, the transformation is order-dependent. In this paper we will investigate three such transformations[2]. The first uses a lexicographic ordering of nodes. We will refer to this as a *lex* ordering. The second, we will call *maxMin*; when selecting a node i to process next we choose a node such that its cheapest incident edge is a maximum over all nodes. For example, in Fig. 1 this would initially select node A or node C, as their cheapest incident edges are largest, with a weight of 4. The intuition behind this is that it will attempt to make the biggest reduction in edge weights. The third ordering is *minMin*. This might be thought of as the anti-heuristic, selecting to process a node i with smallest minimum incident edge weight. In Fig. 1 this would initially choose node B or D.

Order-independent transformation One can also think of a direct transformation that, unlike the ones presented above, does not depend on the order of processing the nodes. Indeed, instead of taking a node, updating its weight and the weights of the incident edges and then choosing a next node, we can do the following:

1. for each node, add a cost of $\delta_i = 2w_{min}$ to w_i, the weight of node i;

[2] In [2] it was shown how to use these transformations in the case of time windows specified on the nodes.

2. after processing all the nodes, for each edge (i, j) subtract a weight of $\frac{\delta_i+\delta_j}{2} = \frac{w'_i - w_i + w'_j - w_j}{2}$ from the cost of edge (i, j), where the prime symbol indicates a transformed value.

This is clearly order-independent. For the same example graph as in Fig. 1 this transformation will result in the graph shown in Fig. 2c. It is easy to check that for this graph, the cost of any cycle remains the same after the transformation. The following theorem generalizes this result.

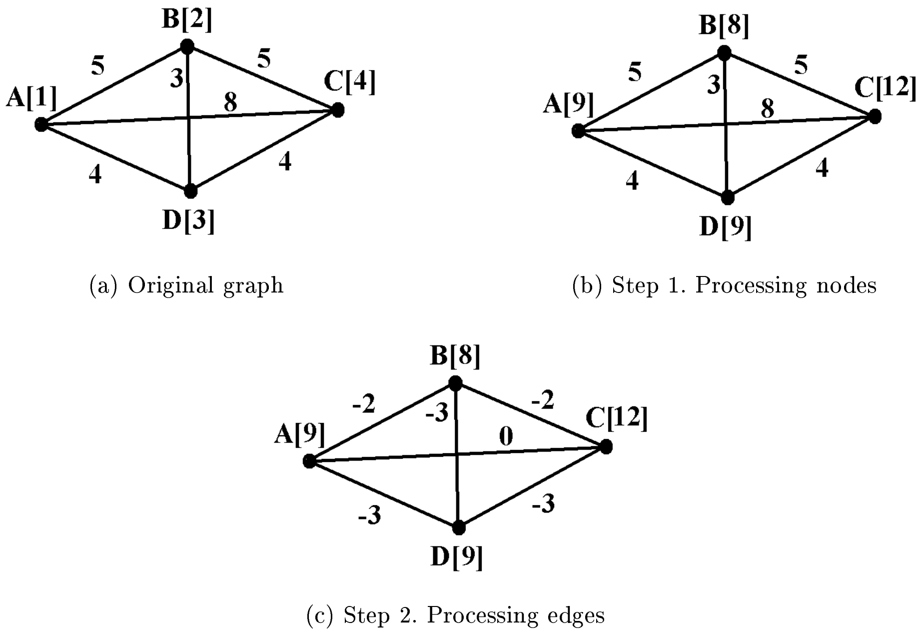


Fig. 2. Order-independent transformation of a 4-node clique. Node weights are shown in square brackets.

Theorem 1. *The order-independent transformation preserves the cost of any solution on an arbitrary complete undirected graph.*

Proof. The proof is by induction. To prove it for $n = 3$ (i.e., for the simplest case of a cycle) we need to express the transformed edge and node weights through the original edge and node weights and substitute these expressions into the cost of the cycle after transformation.

Then assuming $n = 4$ and considering a 4-node tour T we can represent it as two distinct 3-node tours T_1 and T_2 (for which we already know that the transformation preserves the costs) that have a common start (and end) nodes and share an edge. Then we express the cost of T through the costs of T_1 and T_2 bearing in mind that we have to subtract twice the weight of the common edge (because we do not travel along it) as well as the weights of the nodes connected by the common edge (as in T we visit them only once).

Finally, for any $n > 4$ we notice that an n-node tour can be represented as pairs of distinct 3-node tours that, as before, have a common edge and a start node. All the rest can be done as above. *QED.*

If we have a Hamiltonian path, we never enter the start node s. Therefore the amount we add to its weight equals $\delta_s = w_{min}$ instead of $2w_{min}$ as for an arbitrary node. When processing the edges incident on s, we have to subtract a weight of $\delta_s + \frac{\delta_j}{2}$ from the weight of the edge (s, j). The same holds for the end node because we never exit it. All the rest remains as above.

3.2 Inverse transformation

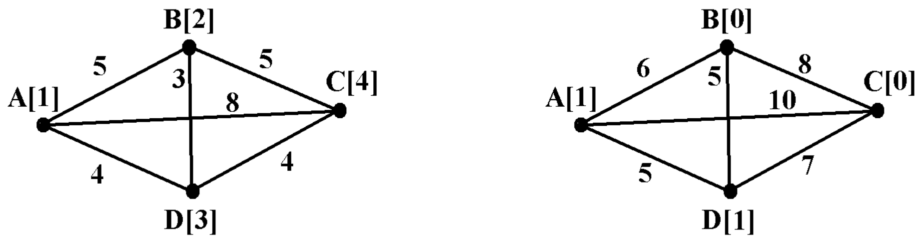
There also exists an *inverse* transformation that reduces node weights by adding a portion of them to the edge weights.

This transformation is as follows. At each step we take an arbitrary node i. Assume, without loss of generality, that the weight of node i is $w_i = 2k+\lambda$, where λ is 0 when w_i is even and 1 otherwise. Observe that we can always subtract $2k$ from w_i and add k to the weight of every edge incident upon i. This will preserve the cost of visiting node i. Indeed, arriving in i along one of the incident edges gives us an additional k of travel, entering node i an additional $-2k$ and leaving the node along another incident edge an additional k).

Observe, first, that this inverse transformation is order-independent. This is because each edge will either increase its weight or retain the original weight in the worst case (in fact, the weight of any edge is updated at most twice, i.e., when processing the two nodes connected by the edge). Second, as a result of this transformation we will have 0/1 node weights. A node weight will be 0 if the original node weight is even, and 1 otherwise.

Consider the same example 4-node clique as above (Fig. 3a). The order of processing the nodes is not important so we assume a lexicographic ordering. Processing node A has no effect as there is nothing to subtract from its original weight of 1 ($w_A = 2k+1 = 1$ and we subtract $2k = 0$). Processing node B results in its weight becoming 0 and the weights of edges AB, DB and CB becoming 6, 4 and 6 respectively. After processing node C its weight becomes equal to 0, the weights of edges AC, BC and DC become 10, 8 and 6. Finally, when processing node D we subtract 2 from its weight (this gives us a weight of 1) and add 1 to the weights of nodes AD, BD and CD (they become equal to 5, 5 and 7). Since, as noted above, we end up having 0/1 node weights, processing a node for a second time will have no effect. The transformed graph is shown in Fig. 3b.

By construction, the inverse transformation always preserves the costs of cycles. The transformation carries over to the case of Hamiltonian paths as follows. For an arbitrary node everything is as above. For the start or end node, we subtract the whole node weight and add it to the weight of each incident



(a) Original graph (b) Transformed graph

Fig. 3. Inverse transformation of a 4-node clique. Node costs are shown in square brackets

edge. As a result, the transformed weights of the start and end node will always be 0 regardless of whether their original weights are odd or even.

4 An Empirical Study

The idea behind our experiments is to study the behavior of the proposed transformations. In [2] it was shown that transformations of the original problem can change the performance of typical JSP and VRP algorithms.

We conjecture that the difference in the performance of these algorithms may be a result of:

1. changes related to the edge weight to node weight ratio (a decrease should favor the JSP techniques);
2. changes in relative edge weights; the relative weights of edges at a node may change and this may serve to disguise real VRP problems and reduce performance in the case of direct transformations or, symmetrically, increase performance in the case of inverse transformations;
3. changes in relative node weights; they might also influence the behavior of different heuristics.

In this section we examine the extent of the changes that the different transformations of sections 3.1 and 3.2 have on randomly generated complete undirected graphs.

The transformations preserve the cost of cycles while changing the weights of edges and nodes. We are interested in the extent to which the weight of a cycle and the weight of the whole graph is transformed to the edges rather than the nodes. To assess this "weight transfer" we have two measurements. The first (Equation (1)) measures the change in the proportion of the weight of a cycle that results from the edge weights. We expect the direct transformations to produce negative values as they reduce edge weights in favor of node weights. In contrast, the inverse transformation should produce a positive value. When measuring ρ_c

for an instance, we generate 1000 random cycles consisting of $|V|$ nodes, where $|V|$ is the number of nodes in the graph, and compute the arithmetic mean $\mu(\rho_c)$.

The second measurement (Equation (2)) assesses the relative change in the relative weight contribution of the edges in the overall graph. This again is a measure of the extent to which the transformations increase the edge weights while decreasing the node weights. As with Equation (1), we expect the direct transformations to result in negative values and the inverse transformation in positive values.

$$\rho_c = \frac{\sum_{(i,j)\in C} w'_{ij} - \sum_{(i,j)\in C} w_{ij}}{\sum_{(i,j)\in C} w_{ij} + \sum_{(i,j)\in C} w_i}, \tag{1}$$

$$\rho = \frac{\frac{\sum_{(i,j)\in E} w'_{ij}}{\sum_{(i,j)\in E} w'_{ij} + \sum_{i\in V} w'_i} - \frac{\sum_{(i,j)\in E} w_{ij}}{\sum_{(i,j)\in E} w_{ij} + \sum_{i\in V} w_i}}{\frac{\sum_{(i,j)\in E} w_{ij}}{\sum_{(i,j)\in E} w_{ij} + \sum_{i\in V} w_i}}. \tag{2}$$

In (1) the summation is for all edges and nodes in a given cycle C. In (2) the summation is for all edges and nodes in a graph; as before, dashed values correspond to the transformed graph.

To estimate the extent of the changes in the relative edge weights we compare all pairs of edges at a node and count the number of times that the weight-based ordering of the pair changes from the original graph to the transformed graph. Similarly to measure the changes in the relative node weights we calculate the number of changes in the relative node weight for all pairs of nodes.

$$\epsilon_{edges} = \frac{\sum \psi'_{ijk}}{N_e}, \tag{3}$$

$$\epsilon_{nodes} = \frac{\sum \xi'_{ij}}{N_n} \tag{4}$$

where $\psi'_{ijk} = 1$ if edges (i,j) and (i,k) at node i change their relative weights as a result of transformation and 0 otherwise; $N_e = |V|\frac{(|V|-1)(|V|-2)}{2}$ is the number of such edge pairs in a graph; $\xi'_{ij} = 1$ if nodes i and j change their relative weight and 0 otherwise; and, finally, $N_n = \frac{|V|(|V|-1)}{2}$ is the overall number of node pairs in the graph. In expression (3) the summation is for all nodes and pairs of their incident edges, in expression (4) for all pairs of nodes.

First, we generate graphs of sizes 10, 20, ..., 100 nodes: 100 instances of each size, 1000 instances in total. Every instance is generated such that the edge weights are uniformly distributed between 1 and 50, the node weights between 0 and 50. To every graph we then apply the five transformations described in sections 3.1 and 3.2.

In Figs. 4 and 5 the arithmetic means $\mu(\rho_c)$ and $\mu(\rho)$ are shown against the graph size. We can see that both $|\mu(\rho_c)|$ and $|\mu(\rho)|$ for the order independent transformation is greater than for any order dependent one. This suggests, therefore, that the order independent transformation may be a better choice when we want to increase an average node weight to edge weight ratio in a graph.

Fig. 4. $\mu(\rho_c)$ vs graph size

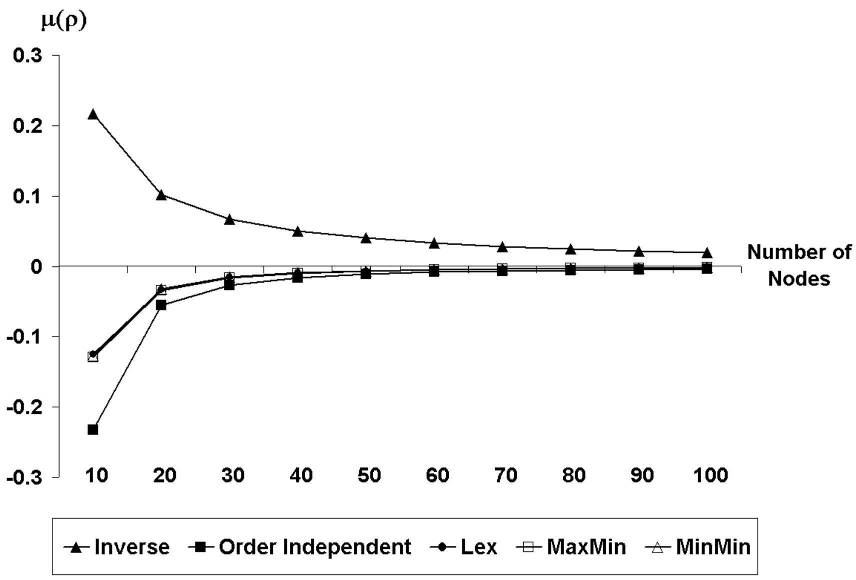


Fig. 5. $\mu(\rho)$ vs graph size

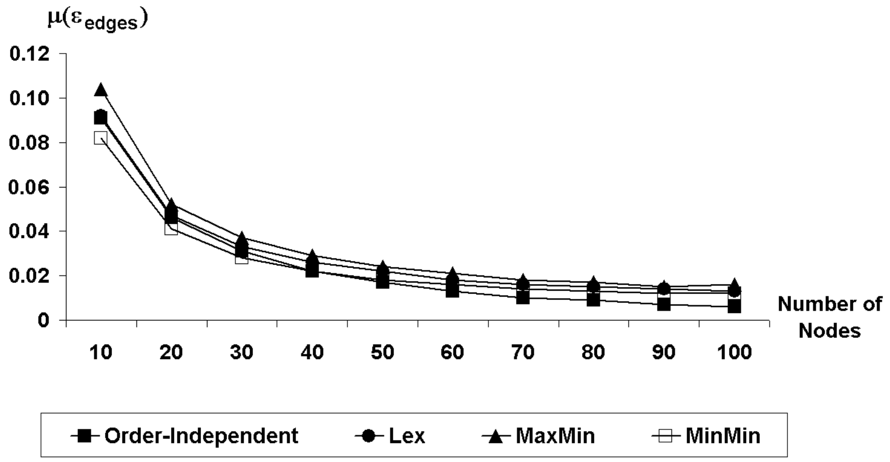


Fig. 6. $\mu(\epsilon_{edges})$ vs graph size

Figs. 6 and 7 depict $\mu(\epsilon_{edges})$ and $\mu(\epsilon_{nodes})$ against the graph size for the direct transformations. Experiments showed that, in contrast to the direct transformations, $\mu(\epsilon_{edges})$ and $\mu(\epsilon_{nodes})$ for the inverse transformation do not depend on the size of graphs. Finally, experimenting with different ranges of edge and node weights in the original generated graphs we obtained results similar to the ones presented above.

Overall, our experiments demonstrate that the order independent direct transformation has a larger impact on the relative edge and node weights than the other direct transformations. That is, the order independent transformation transfers more of the weight of a cycle to the nodes than the other direct transformations. The inverse transformation transfers weights even more strongly as we can see by comparing the absolute values of the inverse transformation to that of the direct transformations.

5 Conclusion and Future Work

The experiments in this paper contribute to a deeper understanding of the graph transformations and allow us to develop some conjectures with respect to scheduling and routing performance. We expect for example, that the direct transformations will tend to increase the performance of scheduling algorithms as they reduce the edge weights in favor of node weights. This translates into a reduction in transition time between activities and longer activity durations. As argued above, this results in a problem that is closer to the problems upon which much of the research in scheduling has been done. Within the direct transformations, we see that the order independent transformation not only transfers more

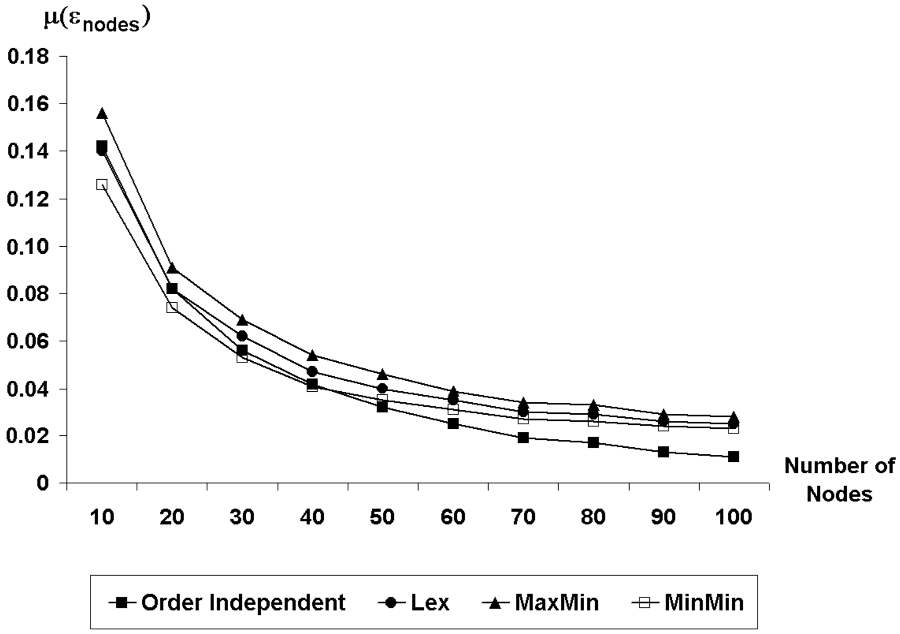


Fig. 7. $\mu(\epsilon_{nodes})$ *vs* graph size

relative weight to the nodes than the other direct transformations but that it does so while performing about the same as the other transformations in terms of relative node weights and relative edge weights. This means that the order independent transformation will not degrade the heuristic ordering of nodes and edges that can be generated from the original problem. As a consequence, we conjecture that the order independent transformation will result in the greatest increase in the performance of scheduling algorithms of all the direct transformations.

In the other direction, the experiments demonstrate that the inverse transformation successfully transforms the graphs to place more weight in the edges of the graphs. We conjecture that this should increase the performance of routing algorithms.

Our future work is to apply these transformations systematically to VRP and JSP problems and to test our conjectures with respect to algorithm performance. In the longer term, we are interested in investigating further transformations in order to isolate the characteristics that contribute to the performance of different search algorithms.

References

1. J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.
2. J.C. Beck, P. Prosser, and Selensky E. On the reformulation of vehicle routing probelms and scheduling problems. APES technical report 44-2002, http://www.dcs.st-and.ac.uk/~apes/apesreports.html.
3. A. Cesta, A. Oddi, and S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 2000. to appear.
4. A.J. Davenport and J.C. Beck. An investigation into two approaches for constraint directed resource allocation and scheduling. In *INFORMS*, 1999.
5. B. DeBacker, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuritics. *Journal of Heuristics*, 6:5001–523, 2000.
6. F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
8. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic, 1995.
9. W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference onf Artificial Intelligence (IJCAI-95)*, pages 607–613, 1995.
10. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
11. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. In *Proceedings of the 6th European Conference on Planning (ECP01)*, 2001.
12. W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.
13. C. La Pape. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.
14. S. Smith and C. Cheng. Slack based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 139–144, 1993.
15. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–365, 1987.
16. C. Voudouris and E.P.K. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):80–110, 1998.