Journal of the Operational Research Society

ISSN: 0160-5682 (Print) 1476-9360 (Online) Journal homepage: https://www.tandfonline.com/loi/tjor20

New heuristic for the dynamic layout problem

E Erel, J B Ghosh & J T Simon

To cite this article: E Erel, J B Ghosh & J T Simon (2003) New heuristic for the dynamic layout problem, Journal of the Operational Research Society, 54:12, 1275-1282, DOI: 10.1057/palgrave.jors.2601646

To link to this article: https://doi.org/10.1057/palgrave.jors.2601646

Published online: 21 Dec 2017.

Journal of the Operational Research Society (2003) 54, 1275–1282

www.palgrave-journals.com/jors

New heuristic for the dynamic layout problem

E Erel[1]*, JB Ghosh[2] and JT Simon[3]

[1]*Faculty of Business Administration, Bilkent University, Turkey;* [2]*Marshall School of Business, University of Southern California, USA; and* [3]*School of Business, State University of New York at Geneseo, USA*

The *dynamic layout problem* addresses the situation where the traffic among the various units within a facility changes over time. Its objective is to determine a layout for each period in a planning horizon such that the total of the flow and the relocation costs is minimized. The problem is computationally very hard and has begun to receive attention only recently. In this paper, we present a new heuristic scheme, based on the idea of viable layouts, which is easy to operationalize. A limited computational study shows that, depending upon how it is implemented, this scheme can be reasonably fast and can yield results that are competitive with those from other available solution methods.
Journal of the Operational Research Society (2003) **54,** 1275–1282. doi:10.1057/palgrave.jors.2601646

Keywords: layout planning; mathematical programming; heuristics

Introduction

Inter-departmental flows account for a significant amount of the cost and the complexity of running a manufacturing/service facility. The basic facility layout problem aims to address this by recommending locations for the various departments such that the resulting flow cost and complexity are minimized. Traditionally, however, this problem has been treated as *static* in the sense that the flows have been assumed to be invariant over time (we thus call it the *static layout problem* or the *SLP*). In today's volatile markets, product life cycles are shrinking, inducing rapid changes in the product mix and volume requirements on a manufacturing facility; volatility and seasonality impose similar changes on a service facility as well. This, in turn, introduces changes in the flow pattern, making an optimal layout for one period less-than-optimal for another. In order to be able to effectively cope with these changes, one needs to be flexible and willing to relocate some of the departments in a manner that is feasible. Unfortunately, relocation is disruptive and can entail a high cost. The basic *dynamic layout problem* (*DLP*) attempts to redress this situation by prescribing a layout for each period in a given planning horizon such that the overall flow and relocation cost is minimized. Variations of the problem incorporate, among others, budgetary constraints, *nervousness* issues arising out of frequent layout changes and planning over a *rolling horizon*.

The basic *SLP* maps naturally to the well-known *quadratic assignment problem* (*QAP*), which is computationally very difficult (*NP-hard*). It has also received substantial coverage in the research literature over an extended period of time.[1] Attention to the *DLP*, in contrast, has come relatively recently and the literature on it to date remains rather sparse. Rosenblatt[2] has been the first to frame the basic problem and sketch out a solution scheme based on dynamic programming. Subsequently, Balakrishnan *et al*[3] have considered a variation involving budget constraints; they have turned to network programming for solution. Urban[4] has proposed a steepest-descent, pairwise-exchange heuristic for the basic problem. Lacksonen and Enscore[5] have attempted to solve the *DLP* by extending the existing solution procedures for the *SLP* (that is, in effect, the *QAP*). Conway and Venkataramanan[6] represent an early effort to apply a generic heuristic (a *genetic algorithm* or *GA* in their case) to the problem's resolution. More recently, Balakrishnan and Cheng[7] through their own *GA* and Baykasoglu and Gindy[8] through a *simulated annealing algorithm* (*SA*) have kept up the thrust in this direction. Balakrishnan and Cheng[9] provide an excellent review of the past work on the *DLP*.

In this paper, we focus on the solution of the basic *DLP* along the lines of Rosenblatt[2] and Balakrishnan *et al*.[3] Following their approach, we plan to arrive at the optimal sequence of layouts by implicitly enumerating over a subset of all possible layouts. Given all possible layouts, the *DLP* can be viewed as a *shortest path problem* (*SP*) on a multi-stage, directed, acyclic network with costs on both nodes and arcs. Each stage corresponds to a time period in the planning horizon, the nodes at any stage represent all possible layouts and the arcs between the nodes in two consecutive stages signify the moves from one layout in one period to possibly another in the next; the node cost is the flow cost of the associated layout in the given time period and the arc cost is the relocation cost between two successive layouts. The above *SP* can be solved exactly via dynamic[2] or network[3] programming. The trouble is that the number of nodes at a

**Correspondence:* *E Erel, Faculty of Business Administration, Bilkent University, Bilkent 06533, Ankara, Turkey.*
E-mail: erel@bilkent.edu.tr

given stage (equal to the number of all possible layouts) is exponentially large. For all practical purposes then, one is forced to work only with a subset of all possible layouts, which one hopes is manageable in size. The generation of this subset is clearly crucial to the effective solution of the *DLP*. It must not be very big and should be obtainable in a short time; it should also contain layouts that are likely to be present in an optimal solution to the *DLP*. Previous suggestions include using random layouts or the k best layouts from each period. We propose a new scheme here in which we consider *viable* layouts that are optimal or quasi-optimal (in terms of the flow cost) with respect to a single period or multiple periods; the idea is somewhat similar to, but much broader than, that used by Urban[4] in a different approach.

In what follows, we first define the problem formally. We then describe the various parts of our solution methodology. The implementation details and the results of a limited computational study are reported next; the results validate that the proposed method can be quite effective for solving the *DLP*. Finally, we conclude with a few closing remarks.

Problem definition

Suppose that we have a facility with N locations where N departments are to be placed, that the planning horizon consists of T periods, and further that the data related to the flow and relocation costs are available for each period in this horizon. Let σ_t be the layout chosen for period t; σ_t can be visualized as an ordered list of the indexes of the departments placed in locations 1 through N. Also, let $C^{\mathrm{f}}(\sigma_t)$ be the flow cost for layout σ_t in period t and $C^{\mathrm{r}}(\sigma_{t-1},\sigma_t)$ be the relocation cost due to the movement from layout σ_{t-1} in period $t-1$ to layout σ_t in period t. In the basic *DLP*, our objective is to find a layout sequence $\{\sigma_1^*,\ldots,\sigma_T^*\}$, which will minimize $\sum_{1\leqslant t\leqslant T} C^{\mathrm{f}}(\sigma_t) + \sum_{2\leqslant t\leqslant T} C^{\mathrm{r}}(\sigma_{t-1},\sigma_t)$

The above formulation is quite general. We narrow it down a bit for our purposes. First, we assume that any department fits into any location; this subsumes the case of equal-sized departments assumed in much of the literature. Then, letting f_{klt} be the volume of the total flow between departments k and l in period t, d_{ij} be the symmetric distance between locations i and j and $\sigma_t(i)$ be the index of the department in location i in period t, we assume that the flow cost for layout σ_t during period t is given by $C^{\mathrm{f}}(\sigma_t) = \sum_{1\leqslant i\leqslant N-1}\sum_{i+1\leqslant j\leqslant N} f_{\sigma_t(i)\sigma_t(j)t}d_{ij}$ (taking, without loss of generality, the cost of unit flow over unit distance to be unity). Similarly, letting m_k be the constant cost of moving department k to a new location in any period t and $\delta_t(i)$ be an indicator variable which is equal to 1 if $\sigma_{t-1}(i)\neq\sigma_t(i)$ and 0 otherwise, we assume that the relocation cost of moving from layout σ_{t-1} in period $t-1$ to layout σ_t in period t is given by $C^{\mathrm{r}}(\sigma_{t-1},\sigma_t) = \sum_{1\leqslant i\leqslant N} m_{\sigma_t(i)}\delta_t(i)$. Finally, we assume that the facility is rectangular and the locations are delineated by equally spaced rows and columns (that are unit distance apart), and also that the distances are measured on the rectilinear scale; thus, if r_i and c_i are, respectively, the row and column indexes of location i, the distance between locations i and j is given by $d_{ij}=|r_i-r_j|+|c_i-c_j|$.

We should note at this point that all of the assumptions stated above are not really necessary for the application of the proposed solution methodology. We have made them in order to be consistent with the past work and for the consequent ease in performing the computational comparisons.

Proposed methodology

In line with earlier work,[2,3] the proposed scheme includes two main phases: the first phase where a *viable* set of layouts is identified, and the second where we implicitly enumerate over this set to solve the *SP* mentioned before. A third phase, seeking local improvement of the solution obtained in the second phase, is also included.

Phase 1: selecting the viable layouts

By a *viable* layout, we mean a layout that is likely to appear in the optimal solution to the *DLP*. As such, we consider layouts that perform the best (in terms of flow cost) with respect to the flow data from a single period or a combination of the flow data from two or more successive periods. In order to obtain these layouts, we first combine the flow data from the T periods using a weighting scheme and then solve the *SLP* (or, synonymously, the *QAP*) for the combined data either exactly or approximately to get the k best layouts. The set of layouts thus obtained is augmented with layouts that are *isomorphic* to (that is, layouts that have the same inter-departmental distances as) the ones in the set (provided such layouts exist) and the augmented set is screened for multiple occurrences of the same layout. The set that we finally get is our *viable* set, Ω. This is the set that is passed on to the second phase for the solution of the associated *SP* problem.

There are thus three steps involved in the generation of Ω: (1) creating a set of weight vectors; (2) combining the T-period flow data using the weight vectors and solving the *SLP* with the combined data for each weight vector to obtain the k best solutions; (3) augmenting the layouts just obtained if possible and screening the resulting set for multiple occurrences.

First Step: Let W be a positive integer. We create the T-period weights $\{w_1,\ldots,w_T\}$ such that w_t, $1\leqslant t\leqslant T$, is 0 or a positive integer and $\sum_{1\leqslant t\leqslant T} w_t = W$. We also ensure that, for any s, t, u such that $1\leqslant s,\ t,\ u\leqslant T$ and $s<t<u$, $w_t\geqslant w_u$ if $w_s>w_t$, and symmetrically, $w_s\leqslant w_t$ if $w_t<w_u$. The rationale is that for $s<t$, if the weight assigned to t is less than that assigned to s, then the periods beyond t should have even

lesser weight. This is essentially an exercise in partitioning the integer W into T parts such that the parts (weights) are non-increasing on both directions from the period(s) with the maximum weight in the T periods. For example, let $W=5$ and $T=5$; in this case, any one of {5, 0, 0, 0, 0}, {0, 1, 3, 1, 0} and {0, 0, 0, 1, 4} will be considered a legitimate weight vector while {0, 2, 1, 2, 0} will not. Let Θ be the set of the weight vectors. The size of Θ dictates the number of *SLP*s to solve in the next step. Thus, while it should be sufficiently large, it should not be too large. The size of Θ is equal to the number of acceptable partitions of W into T parts. This number grows exponentially as a function of W and T. Thus, W along with T determines the size of Θ. In practice, T is likely to be quite small (*viz*, $\leqslant 10$). We recommend using $W=aT$, where a is a parameter under user control and suitably small. We have found using $a \in \{0.5, 1, 2\}$ sufficient for our purposes.

Second Step: Once Θ is determined, for each weight vector $\{w_1,\ldots,w_T\}$ in Θ, we create an instance of the *SLP* with $f_{kl}=\sum_{l\leqslant t\leqslant T} w_t f_{klt}$ for all k, l such that $1\leqslant k\leqslant N-1$, $k+1\leqslant l\leqslant N$. Any instance can be solved using a variety of *QAP* solvers such as the branch and bound algorithm of Burkard and Derigs,[10] the *GRASP* of Resende *et al*[11] and the *GA* of Ahuja *et al*[12] (the first is an exact algorithm whereas the latter two are approximate ones). The idea is to obtain the k best solutions in each case. There are a number of parameters under user control. First, one has to decide whether to solve a *QAP* exactly or approximately. The choice is rather limited here; exact solutions become prohibitively expensive (time-wise) for $N>15$. Secondly, the task of deriving the k best solutions exactly is even more onerous. The approximate algorithms are practical in terms of computational times; the two that we have cited are also of proven quality and can provide the k best solutions without any additional effort. The second parameter has to do with whether to use a single solution or the k best solutions. While more solutions provide diversity, they also increase the size of the set Ω used in the second phase. The third and final parameter deals with the intensity with which to search for an optimal solution to a *QAP*. If time is a concern, the number of node evaluations in the branch and bound algorithm or the number of iterations in the *GRASP* or the *GA* can be limited. After we have solved the *SLP* for each weight vector in Θ according to whatever parameters we have chosen for solving it, we get a set Φ of layouts. Note that a given layout may occur more than once in Φ.

Third Step: A rectangular layout with a row–column configuration has associated with it three other layouts that are images of the original layout or of each other. In that sense, they are *isomorphic* (they have identical interdepartmental distances and thus identical flow costs in every time period). Let {1, 2, 3|4, 5, 6} be a 2×3 layout with the vertical bar separating the rows. {3, 2, 1|6, 5, 4}, {4, 5, 6|1, 2, 3} and {6, 5, 4|3, 2, 1} are its *isomorphic* layouts. For each layout in Φ, we add the three layouts that are *isomorphic* to it. We then screen the expanded set for multiple occurrences, retaining only one occurrence. This yields our viable set of layouts, Ω.

Phase 2: solving the SP over Ω

Given Ω, the *DLP* can be cast as an *SP* on a network (as described before). The *SP* can be solved either via dynamic[2] or network[3] programming. This is something that a user has to decide. We have, however, chosen to use a dynamic programming formulation (*DP*) which relates directly to the network representation.

Let t, $1\leqslant t\leqslant T$, represent a stage, the layout σ_t in Ω represent a state at stage t, and $g_t(\sigma_t)$ be the minimum cumulative cost (flow and relocation combined) up to stage t if σ_t is the layout of choice at that stage. The *DP* recursions are as follows:

For $t=1$:

$$g_1(\sigma_1) = C^{\mathrm{f}}(\sigma_1) \qquad \text{for all } \sigma_1 \in \Omega.$$

For $t=2,\ldots,T$:

$$g_t(\sigma_t) = C^{\mathrm{f}}(\sigma_t) + \min_{\sigma_{t-1}\in\Omega} \{g_{t-1}(\sigma_{t-1}) + C^{\mathrm{r}}(\sigma_{t-1},\sigma_t)\}$$

$$\text{for all } \sigma_t \in \Omega.$$

The optimal value of the total cost can be found from $\min_{\sigma_T\in\Omega}\{g_T(\sigma_T)\}$ and the optimal layout sequence can be constructed through backtracking. The complexity of *DP* is $O(T|\Omega|^2)$.

Phase 3: improving upon the DP solutions

Having solved the *DP*, we can resort to the third phase by picking the k best solutions from Phase 2 and further subjecting each of these solutions to a local improvement procedure. This procedure can be run until a local minimum is reached or a fixed number of iterations has been made. One can conveniently use the same neighbourhood structure as that used by Baykasoglu and Gindy[8] for their *SA*, where a neighbour is obtained by interchanging the locations of two departments within the layout for a given time period.

We have chosen to pursue a simple neighbourhood search scheme. At any iteration, a time period is selected at random, as are two locations. The departments belonging to these locations in the incumbent solution are interchanged to obtain the neighbouring solution. If the neighbouring solution has a total cost lower than the incumbent, it replaces the incumbent. Regardless of what happens, a new iteration is started at this point. We determine the maximum number of iterations as a multiple of the neighbourhood size, which is given by $1/2\cdot T\cdot N\cdot(N-1)$. We have found using a small multiple (specifically 10) to be sufficient.

One last thing to remember is that the *DLP* is a planning problem and thus that the solution time should not be a

major concern in a practice. This makes it possible for a user to experiment with the various parameters until a combination is found that is acceptable in terms of both solution quality and time. The computational study that we report next intends to provide the user some insights that may be useful in carrying out the above task.

Computational study

To test the efficacy of our approach by itself and in comparison to others, we have adopted the test problems furnished by Balakrishnan and Cheng[7] (who have also provided the generation details) and used subsequently by Baykasoglu and Gindy.[8] This set consists of six combinations of N and T ($N=6$, 15, 30 and $T=5$, 10); each combination has eight problem instances, leading to 48 instances in all. For $N=6$ a 2×3 layout, for $N=15$ a 3×5 layout and for $N=30$ a 5×6 layout are assumed.

A few words on the implementation of the proposed solution scheme are now in order. In Phase 1, we have used two values of W, $W=5$ and 10, for the generation of the weight vectors. To keep the experiment manageable, we have settled for the single ($k=1$) best solution to the *SLP* corresponding to a given weight vector (instead of various possible $k>1$ best solutions). We have also exercised the option to invoke or not invoke the improvement phase. For $N=6$, we have used the Burkard–Derigs[10] branch and bound algorithm (the FORTRAN code for which is available in the public domain) to solve the *SLP*; the algorithm has been run to optimality. Depending upon the value of W and whether or not we have invoked the improvement phase, we thus have four implementations here: *DP_5*, *DP_5I*, *DP_10* and *DP_10I*. For $N=15$ and 30, we have similarly used the Resende–Pardalos–Li[11] *GRASP* (the FORTRAN code for which is also available in the public domain); the algorithm has been run for 10 iterations (short mode/S) and 100 iterations (long mode/L). Depending upon the value of W, the number of *GRASP* iterations used and whether or not the improvement phase is invoked, we now have eight implementations: *DP_5S*, *DP_5SI*, *DP_5L*, *DP_5LI*, *DP_10S*, *DP_10SI*, *DP_10L* and *DP_10LI*. As noted before, we use the *DP* in Phase 2.

For performance evaluation purposes, note that the optimal solution values are available for all of the 16 $N=6$ instances and thus that absolute performance of an implementation can be measured. However, this is not true for the 32 $N=15$ and 30 instances; one has to rely here on relative performance only. Fortunately, direct comparisons are possible with the Conway–Venkataramanan[6] *GA* (*GA_CV*), the Balakrishnan–Cheng[7] *GA* (*GA_BC*) and the Baykasoglu–Gindy[8] *SA*; these also represent the most recent computational work on the *DLP*.

As for the two *GA*s, neither the run time information nor the code has been available to us. At any rate, based on our own experimentation and those of others,[7,8] it appears that the *GA*s are generally not competitive. The *SA* due to Baykasoglu and Gindy[8] (*SA_BG*) on the other hand, appears to be quite competitive (for the larger problem instances in particular) and the FORTRAN code has been readily available to us (as part of the Baykasoglu–Gindy paper). However, in an independent experimentation, we have not been able to replicate the reported performance of *SA_BG* for the larger half of the problem set ($N=15/T=10$ and $N=30$).

Baykasoglu and Gindy[8] have set the parameters of *SA_BG* as follows: The initial temperature is determined as $T_{\text{in}}=(f_{\min}-f_{\max})/\ln P_{\text{c}}$, where $f_{\min}$ and $f_{\max}$ are, respectively, the lower and higher bounds on the total cost for a given *DLP* instance (estimated from trial runs), and P_{c} is the acceptance probability at the beginning of *SA*. P_{c} is set to 0.95. The length of a temperature regime *LMC* is set equal to $N\cdot T$. The rate of cooling is set to $\alpha=[\ln P_{\text{c}}/\ln P_{\text{f}}]^{1/(el_{\max}-1)}$, where $el_{\max}$ is the maximum number of iterations and P_{f} is the final acceptance probability. P_{f} is set to 1×10^{-15}. The final temperature can be calculated as $T_{\text{f}}=T_{\text{in}}\alpha^{el_{\max}}$. T_{f} can also be determined as $T_{\text{f}}=(f_{\min}-f_{\max})/\ln P_{\text{f}}$; if these two T_{f} values are not close to each other, then $el_{\max}$ is reselected and and α is recomputed until they are so.

We have chosen to run *SA* on our own. After considering several parameter selection alternatives along the lines of Baykasoglu and Gindy,[8] we have settled for two implementations. In *SA_EG1*, we use a fixed parameter set: initial temperature $T_{\text{in}}=5000$, rate of cooling $\alpha=0.998$, and maximum number of iterations $el_{\max}=5000$ (all other parameters are set as in Baykasoglu–Gindy[8]). In *SA_EG2*, T_{in} and $el_{\max}$ are obtained as in Baykasoglu–Gindy[8] (with $\alpha=0.998$ and final temperature $T_{\text{f}}=1$). In both implementations, five replications are made and the best solutions are noted.

We have coded all algorithms in FORTRAN. All runs have been made on an Ultra Enterprise server operating under Solaris 7 at 250 MHz. Both solution times and values have been recorded. The summary appears in Tables 1–3.

Table 1 shows the results for $N=6$ ($T=5$, 10). In each instance, the optimal solution value as well as the solution values from *DP_10*, *DP_10I*, *DP_5*, *DP_5I*, *GA_CV*, *GA_BC*, *SA_BG*, *SA_EG1* and *SA_EG2* are given (in **bold face** whenever optimal). The mean CPU times are also noted for each block of eight instances with the same T. (Note that the CPU times are not available for *GA_CV* and *GA_BC* and that the CPU times for *SA_BG* come from a different platform.) For $T=5$, *SA_EG2* is clearly the best in terms of solution quality, finding the optimal solutions in all eight cases. For $T=10$, *DP_10* and *DP_10I* are the best, finding the optimal solutions in six out of eight cases. The *DP* algorithms are more than an order of magnitude faster in terms of solution time. The difference between the *DP* solutions themselves is, on an average, less than 0.015% of the optimal. The average gap for *DP_10* and *DP_10I* is

Table 1 Results for $N=6$

			DP solutions				*GA solutions*		*SA solutions*		
T	*Instance*	*Optimal solution*	*DP_10*	*DP_10I*	*DP_5*	*DP_5I*	*GA_CV*	*GA_BC*	*SA_BG*	*SA_EG_1*	*SA_EG_2*
5	1	106 419	**106 419**	**106 419**	**106 419**	**106 419**	108 976	**106 419**	107 249	**106 419**	**106 419**
	2	104 834	**104 834**	**104 834**	**104 834**	**104 834**	105 170	**104 834**	105 170	**104 834**	**104 834**
	3	104 320	**104 320**	**104 320**	**104 320**	**104 320**	104 520	**104 320**	104 800	104 520	**104 320**
	4	106 399	106 509	106 509	106 885	106 515	106 719	106 515	106 515	**106 399**	**106 399**
	5	105 628	**105 628**	**105 628**	105 737	105 737	**105 628**	**105 628**	106 282	105 737	**105 628**
	6	103 985	**103 985**	**103 985**	104 053	104 053	105 606	104 053	**103 985**	**103 985**	**103 985**
	7	106 439	106 447	106 447	106 447	106 447	**106 439**	106 978	106 447	**106 439**	**106 439**
	8	103 771	**103 771**	**103 771**	104 185	104 185	104 485	**103 771**	**103 771**	**103 771**	**103 771**
Mean CPU seconds			<1	<1	<1	<1	**NA**	**NA**	40	55	52
10	1	214 313	**214 313**	**214 313**	**214 313**	**214 313**	218 407	214 397	215 200	**214 313**	**214 313**
	2	212 134	**212 134**	**212 134**	212 138	212 138	215 623	212 138	214 713	**212 134**	213 015
	3	207 987	**207 987**	**207 987**	208 246	208 060	211 028	208 453	208 351	**207 987**	208 351
	4	212 530	212 741	212 741	213 117	212 747	217 493	212 953	213 331	212 747	212 747
	5	210 906	211 022	211 022	211 022	211 022	215 363	211 575	213 812	211 076	211 072
	6	209 932	**209 932**	**209 932**	210 000	210 000	215 564	210 801	211 213	210 000	**209 932**
	7	214 252	**214 252**	**214 252**	**214 252**	**214 252**	220 529	215 685	215 630	214 823	214 438
	8	212 588	**212 588**	**212 588**	213 002	213 002	216 291	214 657	214 513	**212 588**	**212 588**
Mean CPU seconds			<1	<1	<1	<1	**NA**	**NA**	152	215	206

0.016% of the optimal, which is the overall best. The improvement phase for the *DP* implementations has not done anything for $T=10$; for $T=5$, the average improvement has been 0.039%. For $N=6$, among the *DP* implementations, *DP_10I* is the algorithm of our choice.

Table 2 is structured similar to Table 1 and shows the results for $N=15$ ($T=5$, 10). We now have *DP_10L*, *DP_10LI*, *DP_5L*, *DP_5LI*, *DP_10S*, *DP_10SI*, *DP_5S* and *DP_5SI* for the *DP*-based methods. No provably optimal solution being available, we use in each instance the best heuristic solution as our point of reference. *GA_CV* and *GA_BC* are clearly out of the reckoning, with the better of the two having gaps from 4.857 to 10.168% of the best solution. For $T=5$, *SA_EG1* is the clear winner, finding the best solution in all eight cases. *DP_10SI* is the best *DP* implementation, with an average CPU time an order of magnitude faster than the *SA* algorithms and an average gap of 1.102% of the best. For $T=10$, the results reported in Baykasoglu and Gindy[8] for *SA_BG* are the best for all eight cases. *DP_10SI* is once again is the *DP* implementation of choice (based on solution quality and time); its average CPU time is approximately 5 times faster than that of *SA_BG* and its average gap is 3.034% of the best. Overall, the average difference between the various *DP* solutions is less than 0.644% of the best. On an average, the improvement phase reduces the *DP* solution values by a percentage in the range from 0.094 to 0.301.

Table 3 records the results for $N=30$ ($T=5$, 10). The *SA_BG* results are the best for both $T=5$ (finding the best solution in five out of eight cases) and $T=10$ (finding the best solution in seven out of eight cases). As before, *DP_10SI* is the *DP* implementation of our choice (based on solution quality and time). For $T=5$ and 10, its average gap from the best are 1.229 and 2.810%, respectively; the average CPU time is more than an order of magnitude faster than that of *SA_BG* for $T=5$ and 2.75 times for $T=10$. Now, the average difference between the various *DP* solutions is less than 0.571% of the best. The improvement phase reduces the *DP* solution values by an average percentage in the range from 0.154 to 0.293.

In sum, our *DP* implementations have proved to be competitive. It appears that using a long *GRASP* run (100 iterations) does not provide a remarkable advantage over using a short one (10 iterations). The choice of W, however, seems to make a more significant difference ($W=10$ being preferred over $W=5$). The improvement phase appears to be of limited value. Considering solution quality and time, *DP_10I* and *DP_10SI* are the *DP* implementations of our choice for the test bed. However, if time is of concern, one may settle for *DP_5SI*. On an average, for $N=15$ and 30, it produces solutions within 1.378–3.350% of the best solutions and is 12.18–54.84 times faster than *SA_BG*.

Conclusion

In this paper, we have revisited the basic form of the *DLP* and proposed a new solution scheme based on an extension of the early approaches to solving the problem. The proposed scheme is reasonably flexible in that the user can manipulate certain parameters to obtain a desirable balance between solution speed and accuracy. (The exercise of selecting the parameters is quite simple.) Computational results show that this scheme is competitive with the other available solution methods.

Table 2 Results for $N = 15$

			DP solutions								*GA solutions*		*SA solutions*		
T	*Instance*	*Best solution*	*DP_10L*	*DP_10LI*	*DP_5L*	*DP_5LI*	*DP_10S*	*DP_10SI*	*DP_5S*	*DP_5SI*	*GA_CV*	*GA_BC*	*SA_BG*	*SA_EG_1*	*SA_EG_2*
5	1	481 378	484 054	483 568	484 972	482 123	484 369	483 708	484 369	483 708	504 759	511 854	484 695	481 378	481 792
	2	478 816	489 322	489 322	491 102	488 840	487 274	485 702	489 819	488 382	514 718	507 694	486 141	478 816	488 592
	3	487 886	491 310	491 310	493 632	493 632	491 790	491 790	493 224	492 597	516 063	518 461	496 617	487 886	492 536
	4	481 628	487 884	487 275	489 929	489 480	487 956	486 851	489 698	489 698	508 532	514 242	490 869	481 628	485 862
	5	484 177	491 617	491 346	494 040	494 040	491 178	491 178	493 097	491 738	515 599	512 834	491 501	484 177	489 946
	6	482 321	490 205	489 847	490 782	490 782	490 305	489 947	492 275	492 202	509 384	513 763	491 098	482 321	488 452
	7	485 384	490 544	490 051	491 984	490 251	490 161	489 583	492 430	489 155	512 508	512 722	491 350	485 384	487 576
	8	489 072	494 994	493 577	496 841	496 672	494 954	494 534	496 990	496 473	514 839	521 116	496 465	489 072	493 030
Mean CPU seconds			111	119	20	28	14	22	2	10	NA	NA	273	1635	946
10	1	950 910	986 811	984 344	991 093	988 322	986 592	983 070	995 319	991 801	105 5536	104 7596	950 910	982 298	984 013
	2	947 673	985 154	984 779	987 453	985 147	984 601	983 826	988 396	985 360	106 1940	103 7580	947 673	973 179	983 550
	3	968 027	989 081	988 635	993 799	993 318	990 218	990 153	992 824	990 794	107 3603	105 6185	968 027	985 364	988 465
	4	950 701	979 139	976 456	983 208	982 632	978 726	977 548	982 270	982 112	106 0034	102 6789	950 701	974 994	980 045
	5	948 470	986 029	983 846	989 680	985 966	984 975	983 053	987 963	982 893	106 4692	103 3591	948 470	975 498	982 191
	6	948 630	976 917	974 436	979 297	978 683	976 610	975 290	981 406	979 731	106 6370	102 8606	948 630	968 323	973 199
	7	965 844	985 535	982 790	992 897	989 272	987 019	986 325	992 807	988 870	106 6617	104 3823	965 844	977 410	985 270
	8	956 170	990 844	990 372	992 962	988 959	990 247	988 584	993 902	990 376	106 8216	104 8853	956 170	985 041	989 520
Mean CPU seconds			712	724	55	67	206	218	7	19	NA	NA	1042	6470	3867

Table 3 Results for $N = 30$

			DP solutions								*GA solutions*		*SA solutions*		
T	*Instance*	*Best solution*	*DP_10L*	*DP_10LI*	*DP_5L*	*DP_5LI*	*DP_10S*	*DP_10SI*	*DP_5S*	*DP_5SI*	*GA_CV*	*GA_BC*	*SA_BG*	*SA_EG_1*	*SA_EG_2*
5	1	562 405	581 805	579 741	583 082	581 942	581 805	579 741	582 858	581 369	632 737	611 794	562 405	583 081	583 227
	2	569 251	574 657	570 915	576 592	571 563	575 004	570 906	576 106	572 511	647 585	611 873	569 251	573 965	574 116
	3	564 464	581 030	581 030	581 691	580 549	581 170	577 402	581 262	580 186	642 295	611 664	564 464	580 102	577 787
	4	552 684	571 730	569 874	575 024	574 070	571 749	569 596	574 110	573 001	634 626	611 766	552 684	572 139	573 446
	5	559 596	561 079	561 079	561 424	561 424	561 078	561 078	562 857	562 857	639 693	604 564	559 596	563 503	565 735
	6	567 154	567 202	567 154	570 435	570 435	568 554	568 554	570 356	570 356	637 620	606 010	592 515	574 805	570 905
	7	568 196	572 262	568 196	573 878	571 254	572 706	571 580	572 797	569 145	640 482	607 134	582 409	573 361	571 499
	8	575 273	575 445	575 445	576 091	576 091	575 273	575 273	576 149	576 149	635 776	620 183	578 549	581 614	581 966
Mean CPU seconds			1324	1499	222	397	131	306	23	182	NA	NA	3258	21710	10691
10	1	1 122 154	1 174 773	1 171 853	1 180 120	1 171 413	1 172 434	1 171 178	1 181 743	1 180 087	1 362 513	1 228 411	1 122 154	1 175 756	1 174 815
	2	1 120 182	1 175 323	1 169 138	1 179 022	1 174 421	1 175 551	1 170 747	1 177 212	1 170 810	1 379 640	1 231 978	1 120 182	1 173 015	1 177 743
	3	1 125 346	1 174 023	1 174 023	1 175 920	1 170 019	1 175 240	1 165 525	1 176 997	1 173 529	1 365 024	1 231 829	1 125 346	1 166 295	1 171 932
	4	1 120 217	1 155 879	1 152 684	1 157 918	1 156 016	1 155 998	1 153 981	1 158 507	1 156 517	1 367 130	1 227 413	1 120 217	1 154 196	1 154 945
	5	1 128 136	1 128 136	1 128 136	1 131 518	1 131 518	1 129 143	1 128 784	1 132 926	1 132 926	1 356 860	1 215 256	1 158 323	1 141 738	1 140 116
	6	1 111 344	1 144 030	1 143 824	1 147 517	1 147 517	1 144 539	1 144 092	1 149 893	1 149 893	1 372 513	1 221 356	1 111 344	1 158 322	1 158 227
	7	1 128 744	1 143 814	1 142 494	1 147 016	1 145 934	1 143 788	1 143 183	1 147 041	1 146 987	1 382 799	1 212 273	1 128 744	1 157 505	1 163 761
	8	1 136 157	1 168 142	1 167 900	1 170 929	1 170 929	1 167 163	1 167 163	1 171 658	1 171 428	1 383 610	1 245 423	1 136 157	1 179 888	1 177 565
Mean CPU seconds			7008	7358	595	945	1477	1827	63	413	NA	NA	5031	87200	46152

Finally, we note that it is possible to apply the solution framework to other variations of the *DLP*. Budget constraints on single-period relocation costs can be considered simply by prohibiting certain state transitions in the *DP*. A budget constraint on the overall relocation cost can also be accommodated by augmenting the state description with the accumulated relocation cost; an alternative will be to solve the second phase problem by network programming.[3] Certain *nervousness* issues such as the unwillingness to have frequent layout changes (for example, moving to a new layout before the current one has been in place for at least two periods) can similarly be handled by augmenting the state description in the *DP*. Planning over a *rolling horizon* calls for augmenting the state-space possibly with additional layouts (called for by the new flow data) and adding an extra stage to the *DP*.

Acknowledgments—We are grateful to Dr Jaydeep Balakrishnan of the University of Calgary for supplying us with the test problems as well as the optimal solutions to the smaller problem instances.

Addendum

After this paper had been accepted, we learned that the computational results reported by Baykasoglu and Gindy in their paper (Baykasoglu A and Gindy NNZ (2001). A simulated annealing algorithm for dynamic layout problem[8]) were in error. The correct result given in an *Erratum* (Baykasoglu A and Gindy NNZ (2004) Erratum to A simulated annealing algorithm for dynamic layout problem. *Comp Opns Res* **31**: 313–315) show that the algorithms proposed by us in this paper vastly out perform the simulated annealing algorithm of Baykasoglu and Gindy.

References

1 Kusiak A and Heragu SS (1987). The facility layout problem. *Eur J Opl Res* **29**: 229–251.
2 Rosenblatt MJ (1986). The dynamics of plant layout. *Mgmt Sci* **32**: 76–86.
3 Balakrishnan J, Jacobs RF and Venkataramanan MA (1992). Solutions for the constrained dynamic facility layout problem. *Eur J Opl Res* **57**: 280–286.
4 Urban TL (1993). A heuristic for the dynamic facility layout problem. *IIE Trans* **25**: 57–63.
5 Lacksonen TA and Enscore EE (1993). Quadratic assignment algorithms for the dynamic layout problem. *Int J Prod Res* **31**: 503–517.
6 Conway DG and Venkataramanan MA (1994). Genetic search and the dynamic facility layout problem. *Comput Opns Res* **21**: 955–960.
7 Balakrishnan J and Cheng CH (2000). Genetic search and the dynamic layout problem. *Comp Opns Res* **27**: 587–593.
8 Baykasoglu A and Gindy NNZ (2001). A simulated annealing algorithm for dynamic layout problem. *Comput Opns Res* **28**: 1403–1426.
9 Balakrishnan J and Cheng CH (1998). Dynamic layout algorithms: a state-of-the-art survey. *OMEGA* **26**: 507–521.
10 Burkard RE and Derigs U (1980). *Assignment and Matching Problems: Solution Methods with FORTRAN Programs*, Volume 184: Lecture Notes in Economics and Mathematical Systems. Springer: Berlin.
11 Resende MGC, Pardalos PM and Li Y (1996). Algorithm 754: FORTRAN subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Trans Math Software* **22**: 104–118.
12 Ahuja RK, Orlin JB and Tiwari A (2000). A greedy genetic algorithm for the quadratic assignment problem. *Comput Opns Res* **27**: 917–934.

Received January 2002;
accepted September 2002 after two revisions