CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Randomized Local Search Algorithm for the Machine-Part Cell Formation Problem

Khoa Trinh
Jacques A. Ferland
Tien Dinh

June 2011

CIRRELT-2011-35

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Randomized Local Search Algorithm for the Machine-Part Cell Formation Problem

Khoa Trinh[1], Jacques A. Ferland[2,*], Tien Dinh[1]

[1] Faculty of Information Technology, University of Science, VNU-HCMC, 227 Nguyen Van Cu, District 5, Ho Chi Minh City, Vietnam

[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), and Department of Computer Science and Operations Research, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Abstract. In this paper, we study the machine–part cell formation problem. The problem is to assign the given machines and parts into cells so that the grouping efficacy, a measure of autonomy, is maximized. First, we introduce a new randomized local search algorithm, in which the novel part is to solve another subproblem for assigning optimally parts into cells on the basis of given groups of machines. Second, we propose two exact, polynomial–time algorithms to solve this subproblem. Finally, we provide the numerical results of our proposed algorithms, using a popular set of benchmark problems. Comparisons with other recent algorithms in the literature show that our algorithms are able to find high quality solutions while maintaining a very competitive running time.

Keywords. Cell formation problem, fractional programming, randomized local search algorithm.

Acknowledgements. This work was partly supported by the Natural Sciences and Engineering Council of Canada (NSERC) under grant OGP 0008312. This support is gratefully acknowledged.

---

* Corresponding author: JacquesA.Ferland@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2011

1 Introduction

The machine–part cell formation problem (CFP) is an important application of the group technology. The philosophy behind the problem is to decompose an entire manufacturing system, including machines and parts, into many subsystems in such a way that the number of interactions between machines and parts within a subsystem is maximized. In addition, we have to keep the number of interactions between machines and parts belonging to different subsystems as low as possible. By doing so, we reduce the handling cost, the setup cost, and the processing time of operating the whole system [1]. Ogoubi et al. also apply the CFP to design 3–D chip networks [2].

More specifically, in the CFP, we are given an incidence matrix that describes the relationships between machines and parts. We need to group them into several cells so that a certain measure of autonomy is maximized. In fact, there are several different autonomy measures for the problem [3]. In our study, we choose the popular grouping efficacy [4] as the measure of efficiency. Since the CFP is known to be NP–hard [5], most of the methods are based on some heuristics or metaheuristics such as clustering methods [6], genetic algorithms [7], and tabu search [8]. An exact algorithm for this problem would require enormous computational effort, making it virtually impractical [9].

The rest of the paper is organized as follows. First, we formulate the CFP rigorously. We then present a randomized local search algorithm (RLSA). The most novel contribution of this paper is to develop an exact, polynomial-time algorithm for solving a subproblem of assigning parts to the cells once the assignment of machines into cells is known. Solving this subproblem efficiently is important since it has to be executed in each iteration of the RLSA. To this end, we will present two algorithms: one is based on Dynamic Programming (DP) and the other is based on Fractional Programming (FP). Next, we provide the numerical results of our proposed algorithm to solve a set of 35 well–known benchmark problems. Finally, we conclude the paper in the last section.

2 Problem Formulation and Preliminaries

The Cell Formation Problem is often defined as follows. Consider a set $I = \{1, 2, \cdots, m\}$ of machines, and a set $J = \{1, 2, \cdots, n\}$ of parts. An m–by–n incidence matrix A is also available, where $a_{i,j} = 1$ if machine i processes part j, and $a_{i,j} = 0$ otherwise. We refer to a subset of machines as a "machine group." Similarly, we refer to a subset of parts as a "part family." A production cell consists of a pair of a non–empty machine group and a non–empty part family (i.e. in our formulation, we assume that each production cell should contain at least one machine and one part). The objective is to find a solution of K production cells $(C, F) = \{(C_1, F_1), (C_2, F_2), \cdots, (C_K, F_K)\}$ such that $C_1, C_2, \cdots, C_K$ form a partition of I, and $F_1, F_2, \cdots, F_K$ form a partition of J in order to maximize the grouping efficacy:

$$\mathit{Eff}\,(C, F) = \frac{a_1^{In}}{a + a_0^{In}},$$

where

$$a = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j},$$

$$a_0^{In} = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j \in F_k} 1 - a_{i,j},\ a_1^{In} = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j \in F_k} a_{i,j}.$$

Note that a_0^{In} and a_1^{In} are the number of entries equal to 0 and that of entries equal to 1 in all K cells, respectively.

Example 1: Consider the problem with 5 machines and 7 parts where the incidence matrix is the following

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Assuming that we want to find 2 production cells which maximizes the grouping efficacy, one of the optimal solutions should include the cell $(C_1, F_1) = (\{1, 4\}, \{2, 4, 5, 6\})$ and the cell $(C_2, F_2) = (\{2, 3, 5\}, \{1, 3, 7\})$ with the efficacy $\mathit{Eff} = \frac{14}{14+3} \approx 82.35$. Figure 1 illustrates these two cells on the matrix solution.

		Parts						
		2	4	5	6	1	3	7
Machines	1	**1**	**1**	**1**	**1**	0	0	0
	4	**1**	**1**	**0**	**1**	0	0	0
	2	0	0	0	0	**1**	**1**	**0**
	3	0	0	0	0	**1**	**1**	**1**
	5	0	0	0	0	**1**	**0**	**1**

Figure 1: Illustration of the two production cells.

Through out this paper, let $a_1^{In}(j, k)$ be the number of machines in group k that process part j:

$$a_1^{In}(j, k) = \sum_{i \in C_k} a_{i,j},$$

and $a_0^{In}(j, k)$ be the number of machines in group k that do not process part j:

$$a_0^{In}(j, k) = \sum_{i \in C_k} (1 - a_{i,j}).$$

We use $[n]$ to denote the set $\{1, 2, \cdots, n\}$ and let $\mathit{Eff}(C, F)$ be the grouping efficacy of the solution (C, F).

3 A Local Search Algorithm

In this section, we review the local search algorithm (LSA), introduced and experimented by Elbenani et al. (2010). The general idea of LSA is to improve the current solution in each iteration. Starting up with an initial solution, the LSA successively tries to modify the part families (machine groups) on the basis of the machine groups (part families) so that the grouping efficacy increases as much as possible. For this purpose, the authors use a greedy method. In particular, to find new part families $F'_1, \cdots, F'_K$, given the current machine groups $C_1, \cdots, C_K$, for each part j, the authors estimate the approximation of the impact on the grouping efficacy $f(j, k)$ when assigning part j to the family k:

$$f(j, k) = \frac{a_1^{In}(j, k)}{\sum_{i=1}^{m} a_{i,j} + a_0^{In}(j, k)}.$$

Then each part j is assigned to the family $F'_{\tilde{k}(j)}$, where $\tilde{k}(j) = \arg\max_{k \in [K]} f(j, k)$.

If the current solution cannot be improved anymore, then a "destroy&recover" strategy is applied, in which 20% of parts (machines) are swapped in such a way that the grouping efficacy decreases as little as possible. The

LSA stops when the "destroy&recover" procedure is called exactly $5K$ times. Furthermore, whenever the current solution contains some empty group (or family,) a "repair process" is applied to reassign some machines (or parts) inducing the smallest decrease of the grouping efficacy.

Algorithm 1 : Local Search Algorithm

```
Input ← m, n, K, A
Generate an initial feasible solution (C^0, F^0) as in [10]
Initialize (C, F) ← (C^0, F^0), (C_r, F_r) ← (C^0, F^0)
while the destroy&recover procedure is called no more than 5K times do
    Modify with the greedy method the current part families in F on the basis of machine groups in C.
    if there is no improvement then
        Apply destroy&recover strategy to 20% of parts
    if the current solution (C, F) if better than (C_r, F_r) then
        (C_r, F_r) ← (C, F)
    Modify with the greedy method the current machine groups in C on the basis of part families in F.
    if there is no improvement then
        Apply destroy&recover strategy to 20% of machines
    if the current solution (C, F) if better than (C_r, F_r) then
        (C_r, F_r) ← (C, F)
return (C_r, F_r)
```

4 A Randomized Local Search Algorithm

In this section, we propose a randomized local search algorithm based on the LSA mentioned above. Basically, the randomization of the LSA prevents a premature convergence to local optima.

First, we modify the stopping rule, allowing us to improve the current solution for a fixed number of iterations T.

Second, instead of using the greedy method to generate an initial feasible solution, we simply generate an initial solution randomly. In fact, the initial solution is not important compared to the procedure of modifying the current solution in each step.

Third, we change the "destroy&recover" strategy. Whenever we fail to improve the current solution, we simply reassign randomly p percent of parts (machines) to other random families (groups). Overall, the numerical results indicate that the random shuffle strategy seems to be better in that it diversifies the search space and prevents the algorithm from returning to some local extrema. However, we will not present numerical comparisons between these two strategies due to length limit of the paper.

Finally, our main contribution is to introduce a better way to improve the current solution in each iteration. To be precise, we are interested in the exact solution of the following subproblem.

Subproblem 1 *Assume that K fixed machine groups $C_1, C_2, \cdots, C_K$ are specified for the CFP. Determine part families $F_1, F_2, \cdots, F_K$ that form a partition of the set J and maximize the grouping efficacy.*

Proposition 1 *The subproblem 1 can be solved in polynomial–time.*

In the next sections, we will introduce two exact, polynomial–time algorithm to solve the subproblem 1.

Note that the problem of finding machine groups on the basis of specified part families can be solved similarly by working on the transpose of the matrix A. Moreover, in the case that empty families or groups appear in the solution, we have to apply the "repair process" [10] in order to deal with empty families or groups. In summary, the pseudo–code of the RLSA is described as follows.

Algorithm 2 : Randomized Local Search Algorithm

Input $\leftarrow m, n, K, A, T, p$
Randomly generate an initial feasible solution (C^0, F^0)
Initialize $(C, F) \leftarrow (C^0, F^0)$, $(C_r, F_r) \leftarrow (C^0, F^0)$
for $step \leftarrow 1$ **to** T **do**
    Find $F_{opt} = (F_1, F_2, \cdots, F_K)$ on the basis of machine groups in C to maximize the grouping efficacy and update $F \leftarrow F_{opt}$
    if there is no improvement **then**
        Randomly take p percent of parts and reassign them to other random families.
    if the current solution (C, F) if better than (C_r, F_r) **then**
        $(C_r, F_r) \leftarrow (C, F)$
    Find $C_{opt} = (C_1, C_2, \cdots, C_K)$ on the basis of part families in F to maximize the grouping efficacy and update $C \leftarrow C_{opt}$
    if there is no improvement **then**
        Randomly take p percent of machines and reassign them to other random groups.
    if the current solution (C, F) if better than (C_r, F_r) **then**
        $(C_r, F_r) \leftarrow (C, F)$
return (C_r, F_r)

5 A Dynamic–Programming–Based Algorithm For Subproblem 1

5.1 Description

Recall that, for any feasible solution, each part must belong to exactly one of the K families. Let us define an indicator function $f(j, n_0, n_1)$, having the value 1 if and only if it is possible to put the first j parts $\{1, 2, \cdots, j\}$ into the K families $F'_1, F'_2, \cdots, F'_K$ in such a way that

$$n_0 = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j' \in F'_k} 1 - a_{i,j'},$$

and

$$n_1 = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j' \in F'_k} a_{i,j'}.$$

Note that the function f is only defined when $1 \leq n_0, n_1 \leq mn$ and $1 \leq j \leq n$. Since we can only assign the part j into one of the K families, it is easy to verify the following recurrence

$$f(j, n_0, n_1) = \bigvee_{k=1}^{K} f(j-1, n_0 - a_0^{In}(j,k), n_1 - a_1^{In}(j,k)) \tag{1}$$

with $j > 1$, and

$$f(1, n_0, n_1) = 1$$
$$\Leftrightarrow \exists k \in [K] : n_0 = a_0^{In}(j,k) \wedge n_1 = a_1^{In}(j,k).$$

Furthermore, to reconstruct the whole solution, for each reachable state (j, n_0, n_1), we store in $T(j, n_0, n_1)$ one possible choice of family k where the part j can be assigned. After using (1) to compute the values of f for all possible states (j, n_0, n_1), we identify a pair $(\hat{a}_0, \hat{a}_1)$

$$(\hat{a}_0, \hat{a}_1) = \underset{(n_0,n_1)\in[mn]\times[mn]}{\arg\max} \{\frac{n_1}{a+n_0} | f(n, n_0, n_1) = 1\}$$

Indeed, the maximum grouping efficacy is $\frac{\hat{a}_1}{a+\hat{a}_0}$, and we can trace back the complete solution $F_1, F_2, \cdots, F_K$ by using the information recorded in the 3–dimensional array T.

5.2 Pseudo–code

Note that we only need to know the states for which the function f returns 1. To this end, we maintain a queue to keep track of all interesting states for each part j, and we use that information to generate other states when proceeding to part $j+1$.

5.3 Time Complexity Analysis

We can easily precompute the values of $a_1^{In}(j,k)$ and $a_0^{In}(j,k)$ for all $1 \leq j \leq n, 1 \leq k \leq K$ in time $O(mn)$. The total number of states (j, n_0, n_1) is equal to $n(mn)^2 = m^2n^3$. For each state, we need to iterate through all the K possible families to determine the value of $f(j, n_0, n_1)$. Thus, the time complexity of computing the function f for all states is $O(Km^2n^3)$. Finally, finding $(\hat{a}_0, \hat{a}_1)$ and reconstructing the whole solution takes $O(m^2n^2)$. Therefore, the time complexity of the dynamic–programming–based algorithm is $O(Km^2n^2 \max(m,n))$. Practically, this upper–bound is not tight since the number of actual reachable states is much less than $m^2n^2 \max(m,n)$.

Algorithm 3 : Dynamic–Programming–Based Algorithm

Input $\leftarrow m, n, A, K, C$
$f[j, n_0, n_1] \leftarrow 0$ for all $1 \leq j \leq n, 1 \leq n_0, n_1 \leq mn$
Precompute $a_1[j,k] \leftarrow \sum_{i\in C_k} a_{ij}$ and $a_0[j,k] \leftarrow \sum_{i\in C_k} 1 - a_{ij}$ for all $1 \leq j \leq n, 1 \leq k \leq K$
$Q_0 \leftarrow \emptyset$, $a \leftarrow \sum a_{ij}$
for $k \leftarrow 1$ **to** K **do**
    $F_k \leftarrow \emptyset$
    Push $(a_0[1,k], a_1[1,k])$ into the queue Q_0
    $f[1, a_0[1,k], a_1[1,k]] \leftarrow 1$, $T[1, a_0[1,k], a_1[1,k]] \leftarrow k$
for $j \leftarrow 2$ to n **do**
    $Q_1 \leftarrow \emptyset$
    for all $(n_0, n_1) \in Q_0$ **do**
        for $k \leftarrow 1$ to K **do**
            if $f[j, n_0 + a_0[1,k], n_1 + a_1[1,k]] = 0$ **then**
                $f[j, n_0 + a_0[1,k], n_1 + a_1[1,k]] \leftarrow 1$, $T[j, n_0 + a_0[1,k], n_1 + a_1[1,k]] \leftarrow k$
                Push $(n_0 + a_0[1,k], n_1 + a_1[1,k])$ into the queue Q_1
    $Q_0 \leftarrow Q_1$
Find $(\hat{a}_0, \hat{a}_1) \in Q_0$ such that $\frac{\hat{a}_1}{a+\hat{a}_0}$ is maximized, *Eff* $\leftarrow \frac{\hat{a}_1}{a+\hat{a}_0}$
for $j \leftarrow n$ to 1 **do**
    $k \leftarrow T[j, \hat{a}_0, \hat{a}_1]$
    $C_k \leftarrow C_k \cup \{j\}$
    $\hat{a}_0 \leftarrow \hat{a}_0 - a_0[j,k]$, $\hat{a}_1 \leftarrow \hat{a}_1 - a_1[j,k]$
return $(Eff, F = (F_1, F_2, \cdots, F_K))$

6 A Fractional–Programming–Based Algorithm For Subproblem 1

6.1 An Alternative 0–1 Fractional Programming Formulation

In this section, we formulate subproblem 1 as a mathematical optimization problem. First, we introduce the binary variables $y_{j,k}$

$$y_{j,k} = \begin{cases} 1 & \text{if } j \in F_k \\ 0 & \text{otherwise} \end{cases},$$

for all pairs (j,k), $j = 1, 2, \cdots, n$, and $k = 1, 2, \cdots, K$.

Now, for each valid assignment of parts to the K families, we have the relation

$$a_1^{In} = \sum_{j=1}^{n} \sum_{k=1}^{K} \sum_{i \in C_k} a_{i,j} y_{j,k}, \tag{2}$$

and

$$a_0^{In} = \sum_{j=1}^{n} \sum_{k=1}^{K} \sum_{i \in C_k} (1 - a_{i,j}) y_{j,k}. \tag{3}$$

Subproblem 1 can be written as the following 0–1 fractional programming problem M

$$\begin{aligned} \max \quad & \mathit{Eff} = \frac{a_1^{In}}{a + a_0^{In}} \\ \text{subject to} \quad & \sum_{k=1}^{K} y_{j,k} = 1 \qquad\qquad j \in [n] \quad (4) \\ & y_{j,k} = 0 \text{ or } 1 \qquad j \in [n]; k \in [K]. \quad (5) \end{aligned}$$

Note that the constraints (4) and (5) make sure that each part is assigned to exactly one family.

6.2 A Subproblem of Problem M

Let Ω be the set of all feasible solutions y of the program M. In order to solve fractional programming problem M with the Dinkelbach algorithm, we have to solve a sequence of simpler problems, in which the fractional objective function is replaced by a linear function. In particular, the Dinkelbach algorithm specifies a sequence of values $\lambda \in \mathbb{R}$, and the following subproblem $M(\lambda)$ have to be solved.

$$\begin{aligned} \max \quad & \mathit{Eff}(\lambda) = a_1^{In} - \lambda(a + a_0^{In}) \qquad (6) \\ \text{subject to} \quad & y \in \Omega. \end{aligned}$$

Substitute relations (2) and (3) into (6), we have

$$\begin{aligned} \mathit{Eff}(\lambda) &= a_1^{In} - \lambda(a + a_0^{In}) \\ &= -\lambda a + \sum_{j=1}^{n} \sum_{k=1}^{K} \sum_{i \in C_k} a_{i,j} y_{j,k} \\ &\quad -\lambda \sum_{j=1}^{n} \sum_{k=1}^{K} \sum_{i \in C_k} (1 - a_{i,j}) y_{j,k} \\ &= -\lambda a \\ &\quad + \sum_{j=1}^{n} \sum_{k=1}^{K} \left(\sum_{i \in C_k} [(1+\lambda) a_{i,j} - \lambda] \right) y_{j,k}. \end{aligned}$$

Let

$$M_j = \sum_{k=1}^{K} \left(\sum_{i \in C_k} [(1+\lambda)a_{i,j} - \lambda] \right) y_{j,k},$$

we can write

$$\mathit{Eff}(\lambda) = -\lambda a + \sum_{j=1}^{n} M_j.$$

It follows that, to maximize the objective function $\mathit{Eff}(\lambda)$, all separate term M_j's need to be maximized. Due to the constraints (4) and (5), for each part j, all but one value of $y_{j,k}$ are zero, which implies

$$M_{j\max} = \max_{k=1,\cdots,K} \left(\sum_{i \in C_k} [(1+\lambda)a_{i,j} - \lambda] \right).$$

Therefore, we only need to assign each part j to the family $F_{\bar{k}}$ such that

$$\bar{k} = \arg\max_{k=1,\cdots,K} \left(\sum_{i \in C_k} [(1+\lambda)a_{i,j} - \lambda] \right),$$

and the subproblem $M(\lambda)$ is now solved.

6.3 Applying the Dinkelbach's algorithm to solve problem M

The details and analysis of the Dinkelbach algorithm can be found in the work by [12]. The pseudo–code for solving problem M is illustrated as follow.

Algorithm 4 : Fractional–Programming–Based Algorithm

1: Input $\leftarrow m, n, A, K, C$
2: Randomly generate some families $F^0 = (F_1^0, \cdots, F_K^0)$ and initialize $\zeta \leftarrow 0$
3: **repeat**
4: $\quad \zeta \leftarrow \zeta + 1$
5: $\quad \lambda_\zeta \leftarrow \mathit{Eff}(C, F^{\zeta-1})$
6: $\quad$ Solve the subproblem $M(\lambda_\zeta)$ and let F^ζ be an optimal solution.
7: **until** $\mathit{Eff}(\lambda_\zeta) = 0$
8: **return** $(\mathit{Eff}(C, F) = \lambda_\zeta, F^\zeta = (F_1^\zeta, F_2^\zeta, \cdots, F_K^\zeta))$

6.4 Time Complexity Analysis

The algorithm will terminate since the generated sequence $\{\lambda_\zeta\}$ is strictly increasing [12]. Moreover, it can be derived from a past result [11] that our algorithm will only have to solve at most $O(\log(mn))$ subproblems $M(\lambda_\zeta)$. Besides, the time complexity of solving each subproblem $M(\lambda)$ as shown in the previous subsection is $O(nK)$. Therefore, after taking into account the cost of precomputation, the FP–based algorithm runs in time $O(mn + \max(m,n)K\log(mn))$.

7 Experimental Results

In this section, we provide the numerical results of our algorithm as well as comparisons with other state–of–the–art approaches. All algorithms are tested on the 35 well–known benchmark problems for the CFP specified clearly

Table 1: List of all test problems

Instance	Size	K	Best–known solution	Best–known solution Ref	LSA	
					Eff	Time
15	16×30	6	69.53	[10]	66.42	0.27
16	16×43	8	57.53	[13]	56.77	0.49
17	18×24	9	57.73	[13]	53.61	0.35
18	20×20	5	43.17	[10]	41.29	0.20
19	20×23	7	50.81	[15]	50.00	0.30
20	20×35	5	77.91	[13]	76.02	0.33
21	20×35	5	57.98	[13]	56.08	0.33
22	24×40	7	100	[13]	100.00	0.61
23	24×40	7	85.11	[13]	85.11	0.59
24	24×40	7	73.51	[13]	73.51	0.60
25	24×40	11	53.29	[13]	52.94	0.92
26	24×40	12	48.95	[13]	48.28	1.00
27	24×40	12	46.58	[13]	45.58	0.99
28	27×27	5	54.82	[13]	52.09	0.34
29	28×46	10	47.06	[10]	45.49	1.10
30	30×41	14	63.31	[10]	61.33	1.46
31	30×50	13	60.12	[14]	60.12	1.65
32	30×50	14	50.83	[15]	50.56	1.78
33	36×90	17	47.77	[16]	43.83	3.81
34	37×53	3	60.64	[13]	60.34	0.53
35	40×100	10	84.03	[17]	84.03	3.34

in the literature [17, 10]. We have already collected the currently best–known solutions for these instances from various sources [10, 13, 14, 15, 16, 17]. Besides, we compare our RLSA-DP (using DP-based algorithm in RLSA) and RLSA-FP (using FP-based algorithm in RLSA) with another recent algorithm: a hybrid method combining LSA and a genetic algorithm (GA) [10]. Our algorithms will be tested with the parameters $T = 500$ and $p = 10$.

In Table 1, we record the sizes of problems, the best-known solutions as well as the references where the best-known solutions are obtained. In case that the solution is obtained by several different methods, we only mention one reference to that best-known solution. The two last columns indicate the results by the original LSA as well as its running time. Note that, to make it fair, we also increase the number of iterations of the LSA from $5K$ (in all instances: $K \leq 20$) to T.

All algorithms are implemented in C++ using the compiler GCC 3.4.2. We conduct all experiments on a personal computer with the Intel–Quadcore 2.33GHz processor and 4GB RAM. The algorithms GA, RLSA–DP, and RLSA–FP are tested independently for 10 times, after which we record the average *Eff* , the average running–time, and the best solution. The results are summarized in Table 2, but to reduce its size, we only report the results for problems 15 to 35 [10] since the results for other smaller problems are identical for the all methods.

The percentage of gap of the solution by algorithm 1 with respect to the solution by algorithm 2 is measured by the ratio of the difference in those solutions to the solution by algorithm 1.

Analysis: It can be seen in Table 2 that our proposed algorithm RLSA–FP is able to reach the best–known solutions for almost all cases, after 10 independent runs. Remarkably, it manages to improve the best–known solution for two instances: 29 and 33. The average *Eff* of the RLSA–FP is greater than that of the GA for 8 instances with an average gap about 0.4%. For another 5 instances, the GA have better average *Eff* to average gap only about 0.24%.

Table 2: The results of all algorithms

Instance	Best–known	GA			RLSA–DP			RLSA–FP		
	Solution	Max	Avg	Time	Max	Avg	Time	Max	Avg	Time
15	69.53	69.53	69.53	22.14	69.53	69.53	8.50	69.53	69.53	**0.34**
16	57.53	57.53	57.36	87.50	57.53	**57.53**	12.54	57.53	**57.50**	**0.46**
17	57.73	57.73	57.53	52.06	57.73	**57.69**	4.79	57.73	**57.73**	**0.33**
18	43.17	43.17	43.14	18.81	43.17	43.08	4.08	43.17	43.07	**0.27**
19	50.81	50.81	50.81	40.02	50.81	50.81	5.69	50.81	50.81	**0.33**
20	77.91	77.91	77.64	41.97	77.91	**77.91**	13.42	77.91	**77.91**	**0.44**
21	57.98	57.98	57.39	35.81	57.98	**57.98**	11.14	57.98	**57.98**	**0.42**
22	100.00	100.00	100.00	47.16	100.00	100.00	16.47	100.00	100.00	**0.62**
23	85.11	85.11	85.11	78.86	85.11	85.11	16.13	85.11	85.11	**0.61**
24	73.51	73.51	73.51	94.01	73.51	73.51	14.88	73.51	73.51	**0.63**
25	53.29	53.29	53.29	222.95	53.29	53.29	15.17	53.29	53.29	**0.61**
26	48.95	48.95	48.95	296.03	48.95	48.57	11.79	48.95	48.67	**0.64**
27	46.58	46.58	46.58	280.81	46.58	46.34	11.43	46.58	46.37	**0.62**
28	54.82	54.82	54.82	28.01	54.82	54.82	29.06	54.82	54.82	**0.45**
29	47.06	47.06	46.91	328.76	**47.08**	**47.04**	47.63	**47.08**	**47.06**	**0.79**
30	63.31	63.31	63.07	765.03	63.31	**63.20**	25.49	63.31	**63.25**	**0.79**
31	60.12	60.12	60.12	617.88	60.12	60.08	28.01	60.12	60.10	**0.90**
32	50.83	50.83	50.83	838.67	50.83	50.72	28.86	50.83	50.75	**0.96**
33	47.77	47.77	47.74	2199.88	**48.00**	**47.87**	219.16	**48.00**	**47.82**	**1.58**
34	60.64	60.63	60.39	68.68	60.64	**60.64**	203.11	60.64	**60.64**	**1.04**
35	84.03	84.03	84.03	1021.55	84.03	84.03	413.54	84.03	84.03	**2.23**

On average, it takes the RLSA–FP only about 2.3 seconds to solve the largest instance (with a 40x100 incidence matrix) on our computer. The GA runs very slowly on instances that have big values of K. The RLSA–DP requires more computational effort than the RLSA–FP when the size of input increases. Note that, although the original LSA runs as fast as the RLSA–FP, the quality of solutions by this algorithm is not very good. It fails to reach the best–known solution for 21/35 instances.

Indeed, if we increase the parameter T, the average *Eff* of our algorithm will be improved. However, we recommend to keep this parameter small, say, less than 2000, in order to take advantage of its very fast speed while the returned solution is as good as (and sometimes better than) the best solutions found by other current algorithms in the literature.

8 Conclusions

In this work, we have presented a randomized local search algorithm (RLSA) for the cell formation problem. The novel idea is to find a new, better way to improve the current solution in each iteration of the RLSA. To this end, we introduce two exact, polynomial–time algorithms to solve a special case of the CFP, in which an assignment of machines (or parts) is already known. We then experimentally show that our RLSA–FP outperforms other current algorithms in the literature in terms of the quality of solutions as well as the running time. It also improves the best–known solutions for two instances. The time complexity of RLSA–FP is only $O(mn + \max(m, n)K \log(mn))$ if we fix T as a constant, allowing us to solve very large inputs, say, with thousands of machines and parts, in just few minutes on a normal personal computer. This fact suggests further practical use of our algorithm.

Acknowledgements. This research was supported by the NSERC grant (OGP0008312) from CANADA.

References

[1] Wemmerlov U., Hyer N. Research issues in cellular manufacturing, International Journal of Production Research, Volume 25, (1987) 413–431

[2] Ogoubi E., Ferland J.A., Hafid A., Turcotte M., and Bellemare J., A multi–constraint cell formation problem for large scale application decomposition, Working paper, DIRO, Universit de Montral, Qubec, Canada. (2010)

[3] Papaioannou G., Wilson J. M., The evolution of cell formation problem methodologies based on recent studies (1997–2008): Review and directions for future research, European Journal of Operational Research, Volume 206, (2010) 509–521

[4] Sarker B. R., Khan M., A comparison of existing grouping eciency measures and a new weighted grouping eciency measure, IIE Transactions, Volume 33, (2001) 11–27

[5] Dimopoulos C., Zalzala A. M. S., Recent Developments in Evolutionary Computation for Manufacturing Optimization: Problems, Solutions and Comparisons, IEEE Transactions on Evolutionary Computation, Volume 4, Issue 2, (2000) 93–113

[6] Srinivasana G., Narendrana T. T., A nonhierarchical clustering algorithm for group technology, International Journal of Production Research, Volume 29, Issue 3, (1991) 463–478

[7] Tunnukij T., Hicks C., An Enhanced Grouping Genetic Algorithm for solving the cell formation problem, International Journal of Production Research (2010)

[8] Zolfaghari S., Liang M., Comparative study of simulated annealing, genetic algorithms and tabu search for solving binary and comprehensive machine–grouping problems, International Journal of Production Research, Volume 40, Issue 9, (2002) 2141–2158

[9] Elbenani B., and Ferland J.A., An exact method for solving the manufacturing cell formation problem, Publication CIRRELT, Universit de Montral, Qubec, Canada (2010)

[10] Elbenani B., Ferland J.A., and Bellemare J., Genetic algorithm and large neighborhood search to solve the cell formation problem, Publication CIRRELT, Universit de Montral, Qubec, Canada (2010)

[11] Matsui T., Saruwatari Y., and Shigeno M., An Analysis of Dinkelbach's Algorithm for 0–1 Fractional Programming Problems, Dept. Math. Eng. Inf. Phys., University of Tokyo. (1992)

[12] Crouzeix J.P., Ferland J. A., and Hien N., Revisiting Dinkelbach–Type Algorithms for Generalized Fractional Programs, Operational Research Society of India, Volume 45. (2008)

[13] Tunnukij T., Hicks C., An Enhanced Genetic Algorithm for solving the cell formation problem, International Journal of Production research, Volume 47. (2009)

[14] Mahdavi I. , Paydar M.M., Solimanpur M., Heidarzade A., Genetic algorithm approach for solving a cell formation problem in cellular manufacturing, Expert Systems with Applications 36, (2009) 6598–6604

[15] James T.J., Brown E.C., Keeling K.B., A hybrid Grouping Genetic Algorithm for the cell formation problem, Computers & Operations Research 34, (2007) 2059–2079

[16] Luo L., Tang L., A hybrid approach of ordinal optimization and iterated local search for manufacturing cell formation, International Journal of Advance Manufacturing Technology 40, (2009) 362–372

[17] Goncalves J., Resende M.G.C., An evolutionary algorithm for manufacturing cell formation, Computers & Industrial Engineering 47, (2004) 247–273