班级 1704031

学号 17040310084

本科毕业设计（论文）

外文资料翻译

毕业设计题目 《基于蚁群算法的 Petri 网路径规划》

外文资料题目《A hybrid heuristic search algorithm for Scheduling FMS based on Petri net model》

学　　院 机电工程学院

专　　业 自动化

学 生 姓 名 李旭东

指导教师姓名 陈玉峰

《基于 Petri 网模型的调度 FMS 混合启发式搜索算法》

摘要

本文提出了一种在 Petri 网框架中的柔性制造系统（FMS）的新调度方法。Petri 网可以为每个作业，严格的优先级约束，多种资源和并行活动简洁地建模。为了减少拒绝关键标记的可能性，我们的算法对先前生成的标记采用了一种改进的检查方法。为了降低计算复杂度，我们应用了精心设计的方案，该方案在 Petri 网的可达性图中局部执行 A *搜索并全局回溯搜索。我们还进行了一些数值实验，以证明该算法的有效性。

关键字：Petri 网 制造策略 调度 启发式

1、简介

在制造系统中，调度是典型的组合优化问题，它决定开始时间和要处理的作业的分配。一种理想的方法必须包括两个特征：（1）易于解决问题（2）可快速识别半最佳解（只需少量的计算工作）。因此，许多行业和研究社区现在都在致力于开发解决现实世界中的柔性制造系统（FMS）调度问题的方法。但是由于 FMS 调度的复杂性，尚未找到针对所有问题的完美解决方案。一般的 FMS 调度问题属于 NP 组合难题之一[1]，对于此类难题，我们不太可能开发出最优多项式算法。

Petri 网社区最近对 FMS 的性能进行了研究。出于优化目的，选择基于 Petri 网的算法是适当的。尽管 Petri 网作为建模工具非常受欢迎，但由于其状态空间的难处理性，其在优化结果方面并未得到太多关注[2,3]。最近的方法是尝试使用人工智能技术[3，4]通过众所周知的 A*搜索算法（请参阅 Russell 和 Norvig[5]）选择性地搜索 Petri 网可达性图。为最小化系统中零件流动时间的目标函数而开发的在参考文献[2，4，6-9]中提出的 A *搜索算法均源自[2]中提出的原始算法。但是原始算法在某些条件下是不可接受的，例如即使具有可允许的启发式功能它也不能始终保证能得到最优解。孙余等人也提到了这个问题[4]并且他们也为这个问题设计了一种补救措施，但是改进的方法仍然不能保证不拒绝得出最优解的标记[4]。

观察到的另一个问题是难以在合理的时间内找到可观的问题的最佳解决方案，我们引入了一些改进的算法来减少搜索工作。雄和周[8]研究了结合最佳优先(BF)和回溯(BT) 的混合算法的搜索策略。通过引入不可撤销的决策来限制算法回溯功能的 A *算法已在[6,10,11]参考文献中得到实施，一些基于放宽基 A*算法的评估范围的混合搜索算法也被我们考虑在内。

在本文中，我们提出了一种基于时延 Petri 网的混合调度策略。该搜索方案采用：（1）一种针对先前生成的标记的改进的检查方法，以减少拒绝关键标记的可能性和（2）一种比雄和周[8]更精细的方案，该方案在本地执行 BF 搜索，并在全局范围内内执行 BT 搜索。然后通过批量大小不同的示例得出该方法的调度结果，并将其与源自于算法 BF-BT 和 BT-BF 的调度结果进行比较。该算法还适用于一组随机生成的更复杂的 FMS，这些 FMS 具有以下特征：缓冲区大小有限，具有多个资源的操作以及具有备用路由的作业。

2、基于 Petri 网结构的 FMS 模型及调度

Petri 网被定义为包含库所，变迁和连接库所到变迁以及连接变迁到库所的有向弧的双向图。在图形上来说，库所的表示方式是圆圈而变迁的表示方式为条状物或盒型。如果存在将该库所（变迁）与变迁（库所）连接的有向弧，则该库所是一个变迁的的输入（输出）库所。一库所可以包含托肯（即令牌），在图中表示为一个黑点，它可能不包含任何令牌，也可能包含正数的令牌。在任何给定的时间场合上，托肯在库所的分布，称为 Petri 网标识，定义为此网模型系统的当前状态。因此，一个被标记的 Petri 网可用于研究已建立了模型的离散事件系统的动态行为。

事实上，一个 Petri 网[12，13]定义为$Z = (P,T,I,O,m0)$

P $\{p_1,p_2,\dots,p_n\},n > 0$表示库所的有限集合；

T $\{t_1,t_2,\dots,t_s\},s > 0$ *且* $P \cup T = \emptyset$ *和* $P \cap T = \emptyset$表示变迁的有限集合；

I $P \times T \to \{0,1\}$ 代表输入函数或者是从 P 到 T 的有向弧；

O $P \times T \to \{0,1\}$ 代表输出函数或者是从 T 到 P 的有向弧；

m $P \to \{0,1,2 \dots\}$ 代表 P 的模的维度向量，其中 m(p)是变迁 P 的标识计数,m0 是

网系统的初始标识。

为了模拟出一个系统的动态行为，根据下面的变迁发生规则，Petri 网中的一个状态和标识会发生改变。当$m(p) \geq I(p,y)$*且任意的*$p \in P$那么一个变迁就有了发生权。一个具有发生权的变迁在标识m'处触发，并且其触发会产生一个新的标识，$m(\mathrm{p}) = \mathrm{m}'(\mathrm{p}) + \mathrm{O}(\mathrm{p},\mathrm{t}) - \mathrm{I}(\mathrm{p},\mathrm{t})$,其中 p 是任意的且来自 P。

标记 m 被认为可以从 m’ 出发并到达。给定 Z 及其初始标记 m0，可达集是从 m0 通过变迁触发的各种序列可到达的所有标记的集合，并用 R（Z，m0）表示。对于标识 m∈R（Z，m0），如果在 m 中未触发任何变迁，则 m 被称为死锁标识，并且相应的系统处于死锁状态。

在本文中，我们假设所有作业一开始都是可用的，并且没有新作业到达系统。对于这些假设，我们遵循参考文献[2，8，11]中介绍的建模方法。库所表示资源状态或操作，变迁表示事件或操作过程的开始或完成，一个活动的停止变迁将与之后的下一个活动的开始变迁相同。资源库所中的托肯表示该资源可用，操作库所中的托肯表示该操作正在执行，没有令牌表示未执行任何操作。在操作开始和结束之间可能要经过一定的时间，通过将时间信息与相应的操作库所相关联来表示。

在一个时延 Petri 网框架中搜索事件驱动的时间表，以实现最小或接近最小的制造时间。本文通过将时间延迟与库所相关联来使用一种确定型的时延 Petri 网。可以以零时长触发变迁，这与非时延 Petri 网的定义一致。在一个系统的 Petri 网模型中，触发一个有发生权的变迁会改变托肯分布（标识）。一连串的触发会产生一系列的标识，并且系统的所有可能行为都可以通过网络的可达性图完全跟踪。最优事件序列的搜索空间是这个网的可达图，我们的问题是要找到 Petri 网模型中从初始标识到最终标识的触发序列。Lee 和 Dicesare [2]首先引入了可应用于 Petri 网的 A *算法，A *搜索是一种预知的搜索算法，它仅扩展一个时延库所 Petri 网的可达性图的最期望的分支，基本算法[2]如下：

算法 I(A*算法)：

1 将初始标识 m0 放到列表 OPEN 中。

2 如果 OPEN 这个表是空的，最终结果会失败。

3 从 OPEN 中移走第一个标识 m 并且将这个 m 移入到列表 CLOSED。

4 如果 m 是最终标识，则构造从初始标识到最终标识的最优路径并终止。

5 找到标识 m 的具有发生权的变迁。

6 为每个具有发生权的变迁生成下一个标识或后继标识，并给从下一个标识到 m 的这个过程设置指针，为每个后继 m’计算 g（m’）。

7 对于 m 的每个后继 m’：

a 如果 m’已经在 OPEN 表中，将其指针沿路径引导，产生最小的 g（m’）。

b 如果 m’已经在 CLOSED 表中，则将其指针沿路径引导，产生最小的 g（m’）。 如果 m’需要指针重新定向，则将 m’移至 OPEN。

c 计算 h（m’）和 f（m’），然后将 m’置于 OPEN 位置。

8 以 f 的递增量对 OPEN 列表重新排序。

9 转到步骤 2。

根据以下表达式计算算法 1 中的函数$f(m)=g(m)+h(m)$。$g(m)$表示到目前为止确定的部分调度表的最大完工时间。另一方面，称为启发式函数的$h(m)$表示对达到表示目标状态m_f的标识的剩余花费时间（即最大完工时间）的估计。启发函数的目的是通过建议先触发哪种变迁来引导搜索过程朝最有利的方向发展。

如果$h(m)$是从当前标识后的所有完整解的下界，即：

$$h(m)\leq h^{*}(m),\forall m \tag{1}$$

其中$h^{*}(m)$是从当前标识 m 到最后一个标识的路径的最优化成本，$h(m)$是被允许的，这保证了网系统能够得到最佳解决方案[14]。

在 A *搜索过程的每个步骤中，选择到目前为止生成的最有希望的标记，这是通过对每个对象应用适当的启发式函数来完成的。然后它通过在该标识下触发所有具有发生权的变迁来扩展所选标识，如果后续标识之一是最终标识，则算法退出；如果不是，则将所有这些新标识添加到到目前为止生成的标识集中。然后再次选择最有希望生成的标识，然后继续该过程。一旦构建了系统的 Petri 网模型，并给出了初始标识和最终标识，便可以使用具有允许启发算法共能的上述算法[2]获得最优调度。

3、检查先前生成的标识

在上述搜索过程中，当获得新标识时，作为当前搜索节点的后继，可以将其与该点之前生成的所有标识进行比较。相同的标记可能表示达到相同状态的不同路径，因此我们必须检查条件以确定新路径是否比现有路径具有更大的期望度。在算法 1 中，比较标识和当前最大完工时间的简单测试不能满足此要求，因此可能会使算法拒绝使网系统达到最优解的路径[15]。

孙禹等人也认识到了这个问题[4]并为这类问题设计了一种补救方法，当比较标识时用的是以下的可选式最大完工时间花销函数来取代$g(.)$：

$$j(M) = g(M) + \sum t_i / |U_M| \tag{2}$$

其中$\sum t_i$是不可用托肯的剩余时间与的总和。$|U_M|$是不可用托肯的数量（某个位置中不可用的托肯意味着尽管该令牌尚未准备好，但仍将在固定时间后变为可用的）。然而它仍然不能保证不拒绝达到最佳解决方案的标识[4]。

为了解决这些问题，我们必须了解时延 Petri 网状态表示的过程。由于时间延迟的概念与库所相关联，因此托肯可以具有两种可能的状态。库所 p 中的托肯有两种可能的意义：（1）对于以 p 为输入库所的任何变迁，此托肯都可以被视为输入令牌，或者（2）尽管该托肯尚未准备好，但在固定时间之后它将会变为可用的（通常发生在操作库所）。因此在第二种情况下，这种托肯会有一段剩余处理时间(t_l)才会变为可用的。随着时间的流逝，t_l减小直至达到零，当t_l达到零时，这些托肯变为可用。因此，可能需要额外的时间t_l才能实际“到达”标识 m，因此在孙禹等人提出的算法 1 中步骤 7a 的路径更新条件$m' = m^O \wedge g(m') < g(m^O)$（或者步骤 7b 中的$m' = m^C \wedge g(m') < g(m^C)$）和$m' = m^O \wedge j(m') < j(m^O)$都不能预示出一个从标识$m_0$到标识$m'$的新的更好的路径（$m_0$是初始标识，$m^O$和$m^C$分别代表列表 OPEN 和 CLOSED 中的一个标识）。在本节中，我们提出了一种替代检查方法，该方法同时考虑标识和标识的剩余处理时间以选择更好的节点。为了表示出我们的方法，我们需要对延时标识下定义。

定义 1：时延 Petri 网的时延标识 m 为三元组(M, Mr, g),其中：

- M 是 Petri 网中可用托肯和不可用托肯的标记；
- Mr 是一个向量，其中包含每个库所中托肯的剩余处理时间；

• g 是从 M0 到达 M 的路径花费成本。

我们使用M_m，Mr_m和$g(M_m)$表示一个基于时间的标识 m 的元素。 现在让我们考虑时延 Petri 网的三个状态，并且在这三种状态下 M 相同（图 1）。 然后，如果$g(M_m) < g(M^O)$并且每一个Mr_m的值都不大于每一个Mr^O的值，则可以确定找到了M_0和M_m之间的更好路径（最后一个状态表示为$Mr_m \leqslant Mr^O$）。这在算法 2 中进行了描述。

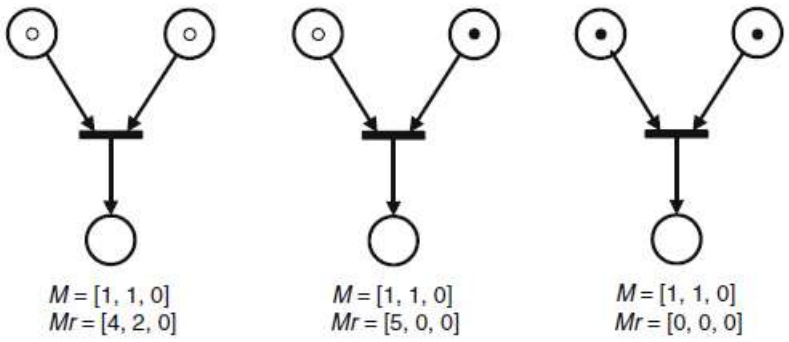

图 1 变迁触发的描述

算法 2（增强后的 A*算法）

1 将初始标识 m0 放置在 OPEN 列表中。

2 如果 OPEN 为空，则以失败终止。

3 从 OPEN 列表中去掉第一个标识 m，然后将这个 m 放置在 CLOSED 列表中。

4 如果 m 是最终标记m_f，则构造从 m 返回 m0 的最佳路径并终止。

5 找到一组有发生权的变迁$\{t_j\}$ $(j = 1 \ldots et(m))$。其中$et(m)$是当标识为 m 时具有发生权的变迁的数量。

6 生成子标识m'_j，而这些子标识正是由每个有发生权的变迁t_j触发而产生，然后计算$g(m'_j)$，$h(m'_j)$和$f(m'_j)$

7 对每个标识M'_j，做以下操作

a 如果标识m'_j和已经在 OPEN 列表中的一些标识m^O等同，比较Mr和 g，如果有$Mr'_j \geq Mr^O \wedge g(m'_j) \geq g(m^O)$，那么跳转到步骤 2；如果$Mr'_j \leq Mr^O \wedge g(m'_j) < g(m^O)$,那么从 OPEN 列表中删除标识$m^O$，并插入$m'_j$进入此列表中。如果以上两

种情况都不满足，那么直接在 OPEN 列表中插入m_j'。

b 如果m_j'和已经在 CLOSED 列表中的一些标识m^C等同，比较Mr和 g,，如果有$Mr_j' \geq Mr^C \wedge g(m_j') \geq g(m^C)$,那么跳转到步骤 2; 如果$Mr_j' \leq Mr^C \wedge g(m_j') < g(m^C)$,那么从 CLOSED 列表中删除标识$m^C$，并插入$m_j'$到 OPEN 列表中。如果以上两种情况都不满足，那么直接在 OPEN 列表中插入m_j'。

8 以 f 的递增量对 OPEN 列表重新排序。

9 转到步骤 2。

在该算法中，当一个标识生成时，我们将其 M，Mr 和 g 的值与已经生成的标识的这些值进行比较。如果它比 OPEN 或 CLOSED 上已有的标识好，那么我们将其插入 OPEN 列表并删除 OPEN 或 CLOSED 上的标识。如果情况更糟，我们将直接拒绝插入列表中。如果无法暂时评估标识的品质，那么我们先将次标识插入 OPEN 中，该方法对 Lee 和 DiCesare[2]引入的基本算法进行了修改，并且它不会拒绝有希望的标识，其他细节可以在[15]中看到。

4 A*-BT 结合算法

A *搜索是最优的，并且是最有效的。但是对于量大的调度问题，这种搜索算法很难在合理的时间内找到最优解决方案，而且在许多问题中，A *算法需要花费大量时间来区分路径花费成本互不相同的路径[14]。

为减少纯 A *算法策略所需的记忆空间和计算时间，雄和周[8]提出了两种混合搜索方法（BF-BT 和 BT-BF），它们结合了启发式 A *算法策略和基于执行 Petri 网的 BT 算法策略。BF-BT 的算法描述在图 2a 中，其中 A *策略应用于搜索图的顶部（由具有不规则边界的阴影区域表示），而 BT 策略则应用于底部（由左至右的箭头表示）。一旦到达深度界限d_0，BT 搜索就会从 OPEN 列表上的最优节点开始遍历直到该节点下方的整个图形都遍历完。在图 2b 中描述了另一种算法 BT-BF，此处 BT 策略应用于图表的顶部，而 A *策略则应用于底部，直到达到深度界限d_0采用 BT 策略，在此点上搜索策略不会倒退，而是从d_0处的节点开始 A *搜索，直到它返回解决方案或存在故障为止。

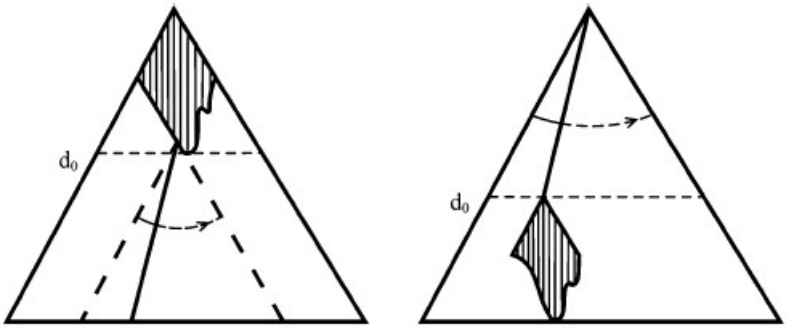

图 2 左图表示在顶部（阴影部分）先做 BF 搜索，然后再做 BT 搜索结尾，右图表示从 BT 开始搜索（从左至右的箭头），然后做 BF 搜索结尾

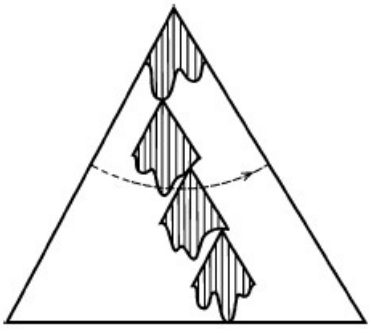

图 3 对系统局部实施 BF 搜索，对全局实施 BT 搜索

因为 BF 搜索的性能通常在搜索图的底部处于最佳状态，所以 BT-BF 的性能要好于 BF-BT [8]。但是 BT-BF 算法存在两个主要缺点：（1）关于调度质量的重要决策可能会在调度活动的早期阶段发生[8]；这增加了错过 BT-BF 搜索到关键候选对象的可能性，值得一提的是 BT-BF 搜索在早期使用 BT 搜索而不是 BF 搜索。（2）解决方案的深度通常不是先验的，因此 BT-BF 策略很难决定在何处标识触发从 BT 到 BF 的变迁是正确的。

本节中提出了一种更详尽的方案，该方案在局部执行 BF 算法，在全局范围内执行 BT 算法。如图 3 所示，我们开始以 A *方式搜索，在搜索过程的每个循环中，在从父节点生成了所有后继节点之后我们检查 OPEN 列表上的节点数，一旦此数量超过阈值M_{max}，我们将 OPEN 上的所有节点视为根节点的直接后继丛，并将其提交给 BT 搜索。BT 搜索在这些后继中选出最好的，并使用 BF 搜索算法“扩展”这个后继，也就是说它将选定的节点交给局部的 A *搜索算法，直到 OPEN 上的节点数再次超过阈值M_{max}，然后将 OPEN 上的新节点视为节点“扩展”后的直接后继丛，该策略在算法 3 中得到实施。

算法 3（局部 A*算法和全局 BT 算法结合）

1 $i = 0$;

2 将初始标记m_0放置在列表$OPEN_i$中。

3 如果$i > 0$，则从列表$OPEN_{i-1}$中删除第一个标记 m 并将 m 放在列表$OPEN_i$中

4 如果$OPEN_i$为空，则检查 i，如果$i = 0$，则此算法以失败终止；如果$i > 0$则转到步骤 3。

5 从列表$OPEN_i$中删除第一个标记 m，然后将 m 放在列表$CLOSED_i$中。

6 如果 m 是最后的标记m_f，则构造从 m 到m_0的路径并终止算法。

7 找到一组有发生权的变迁$\{t_j\}$ $(j = 1 \ldots et(m))$。其中$et(m)$是当标识为 m 时具有发生权的变迁的数量。

8 生成子标识m_j'，而这些子标识正是由每个有发生权的变迁t_j触发而产生，然后计算$g(m_j')$，$h(m_j')$和$f(m_j')$

9 对每个标识M_j'，做以下操作

a 如果标识m_j'和已经在 OPEN 列表中的一些标识m^O等同，比较Mr和 g，如果有$Mr_j' \geq Mr^O \wedge g(m_j') \geq g(m^O)$，那么跳转到步骤 4；如果$Mr_j' \leq Mr^O \wedge g(m_j') < g(m^O)$,那么从 OPEN 列表中删除标识$m^O$，并插入$m_j'$进入此列表中。如果以上两种情况都不满足，那么直接在 OPEN 列表中插入m_j'。

b 如果m_j'和已经在 CLOSED 列表中的一些标识m^C等同，比较Mr和 g,，如果有$Mr_j' \geq Mr^C \wedge g(m_j') \geq g(m^C)$,那么跳转到步骤 4; 如果$Mr_j' \leq Mr^C \wedge g(m_j') < g(m^C)$,那么从 CLOSED 列表中删除标识$m^C$，并插入$m_j'$到 OPEN 列表中。如果以上两种情况都不满足，那么直接在 OPEN 列表中插入m_j'。

10 以 f 的递增量对 OPEN 列表重新排序。

11 如果列表$OPEN_i$中的节点数量超过阈值M_{max}，然后让$i = i + 1$并且转到步骤 3，否则跳转到步骤 4

总而言之，此策略等相当于运行一个预知的深度优先搜索，此搜索每个节

点的扩展都是通过内存受限的 A *搜索完成，并且在 OPEN 列表上的节点被定义为子节点。

对于雄和周[8]中的示例，测试了三组大小为（5、5、2、2），（8、8、4、4）和（10、10、6、6）的数据，我们采用的算法 3 具有与雄和周[8]中相同的可允许启发式函数$h(m)$，该代码全部用 C＃编写，测试是在具有 AMD Athlon 微处理器的个人计算机上以 1 GHz 的时钟频率以及 512 MB 的内存进行的。表 1、2 和 3 中显示了批量大小分别为（5、5、2、2），（8、8、4、4）和（10）的最大完工时间的调度结果，生成标识的数量和计算时间 ，10、6、6）。 这些表格中还显示了从 A *搜索，BT 方法以及熊和周[8]中的 BF-BT 和 BT-BF 算法获得的不同例子的结果。

Algorithm	Depth of BF orM_{max}	Makespan	Number of markings	CPU time (s)
A*		58	3,437	14
BT		105	85	571
BF-BT	20	94	571	0.65
	40	85	1,607	4
	50	79	2,123	6
	60	74	2,775	8
	80	64	3,308	11
BT-BF	20	88	248	0.38
	40	80	484	0.8
	50	70	1,247	3.6
	60	64	1,520	6.5
	80	62	1,687	7
A* locally and BT globally	1	75	230	0.17
	5	68	358	0.31
	10	67	634	0.62
	15	65	540	0.46
	20	62	946	0.78
	25	62	921	0.93
	30	60	1,423	1.12

表 1 批量示例（5，5，2，2）的调度结果

Algorithm	Depth of BF or M_{max}	Makespan	Number of markings	CPU time (s)
A*		100	9,438	112
BT		198	145	0.23
BF-BT	40	168	3,888	24
	60	154	5,234	38
	80	140	7,699	49
	100	127	8,819	90
	120	108	9,233	104
BT-BF	40	163	585	1.4
	60	140	1,590	7
	80	121	2,873	18
	100	112	4,545	36
	120	104	8,045	76
A* locally and BT globally	1	128	422	0.31
	5	116	570	0.46
	10	111	782	0.62
	15	112	1,047	1.56
	20	109	1,694	2.65
	25	105	1,585	2.03
	30	105	1,780	2.65

表 2 批量实例（8，8，4，4）的调度结果

Algorithm	Depth of BF or M_{max}	Makespan	Number of markings	CPU time (s)
A*		134	23,092	720
BT		274	193	0.38
BF-BT	80	206	6,281	64
	100	198	12,341	240
	120	180	16,602	480
	140	169	20,155	540
	160	153	21,797	660
BT-BF	80	209	1,254	5
	100	181	2,315	16
	120	162	8,495	139
	140	150	11,368	390
	160	148	18,875	560
A* locally and BT globally	1	170	581	0.62
	5	155	809	0.93
	10	146	1,089	1.25
	15	147	1,538	2.50
	20	144	1,889	2.81
	25	142	2,850	5.62
	30	141	2,995	7.03

表 3 批量实例（10，10，6，6）的调度结果

所有 BF-BT 算法，BT-BF 算法以及算法 3 都通过缩小评估范围而降低了计算复杂性，但这是以失去最优性为代价的。，图 4、5 和 6 中分别显示了三种不同批量大小（5、5、2、2），（8、8、4、4）和 （10、10、6、6）数据的减小生成标识的数目和最优性损失的关系。 在这些图中，最优性损失的百分比即最大完工时间的比较等同于：

$$RDms = \frac{ms(Hybrid) - ms(A^*)}{ms(A^*)} \times 100\% \quad (3)$$

降低的计算复杂性百分比，即存储空间的比较（生成标识的数量）等于：

$$RDG \frac{GM(A^*) - GM(Hybrid)}{GM(A^*)} \times 100\% \quad （4）$$

从测试的结果我们可以得出下面的结论。

在 Petri 网可达性图中（算法 3）在局部采用 A *算法并全局采用 BT 算法的混合启发式搜索算法的性能要比算法 BF-BT 和 BT-BF 更好，这有三个原因。首先启发式 BF 搜索的性能在其指导启发式信息预知性更好的情况下处于最佳状态，这通常发生在搜索图的底部[14]。因此与仅在搜索图的顶部采用 A *搜索的 BF-BT 搜索相比，在搜索过程的所有阶段局部采用 A *搜索（BF 搜索）的算法 3 大大降低了计算复杂度；其次关于调度质量的重要决策可能会在调度活动的早期阶段发生[8]，因此，与早期采用 BT 搜索而不是 A *搜索的 BT-BF 搜索相比，算法 3 可以更大程度地降低错过关键候选标识的可能性；第三，算法 3 对先前生成的标识采用了改进的检查方法，进一步降低了拒绝关键标识的可能性。当$M_{max} = 1$时，算法 3 不仅不会退化为 BT 方法，而且仍然可以胜过 BT 方法，这是由于以下事实：这两种算法对节点生成采用了不同的策略，BT 方法采用后进先出策略进行节点生成，当首先选择一个标识进行扩展时，将仅触发具有发生权的变迁中的一个，从而仅生成其后继标识中的一个，这个新生成的标识再次被提交以进行扩展。然而在算法 3 中当$M_{max} = 1$时，在一个标识得以扩展后，列表 OPEN 中的最佳标记被选择用于下一次扩展，因此在算法 3 的每个步骤中以$M_{max} = 1$扩展的标识的质量（恰好最终的标识代表解决方案已找到）要优于 BT 方法的扩展标识的质量。

5 应用于更复杂的例子

在本节中，我们将用更复杂的问题对算法 3 进行测试，这些问题具有以下特征：（1）大小有限的缓冲区（2）具有可选路径的作业（3）具有多种资源的运算。我们在图 7 中对这些特性进行了说明，其中$O_{i,j,k}$表示使用第 k 个资源处理过的第 i 个作业类型的第 j 个操作。当缓冲区大小有限时，可以如图 7a 所示修改模型，缓冲区大小由托肯数量表示，例如图 7a 中所示模型的缓冲区大小为 3。在图 7b 中作业 i 的第 j 个操作可以通过可选路径来执行，也就是说通过使用资源 k 或资源 r 来执行。在图 7c 中，操作的执行需要多个资源，即资源 k 和资源 r。

我们生成了一组 40 个随机问题来测试该算法，这些问题是通过随机选择和预定义连接的 Petri 网模块[11]的方法产生的，具有以下特征：每个系统具有三个资源、四个不同的作业和三个操作，75%的工作具有两个可选路径，而 40%的操作具有双重资源，每个操作分配了均匀分布（1 到 100）中的随机路径花销，每个缓冲区的大小限制为 1 到 3，每个作业的批量大小也为 1 到 3。

我们使用算法 3 解决了相同的问题集，其M_{max}值为[1，5，10，15…60]，并且我们将这些问题集与当M_{max}设置为无穷大(纯 A*策略)时得到的结果进行比较，调度结果我们总结在图 8 中，最上面的曲线代表最优性损失的平均百分比(RDms)，中间的曲线代表搜索努力减少的平均百分比(RDGM)，最下面的曲线代表计算时间减少的平均百分比(RDtime，它是计算时间的比较，等于$RDtime = \frac{A*-Hybrid}{A*} \times 100\%$)。我们可以看到算法 3 可以大大减少计算复杂度(生成标识的数量和计算时间)，并且平均最优性损失相当低，例如对于$M_{max} = 30$，平均来说，它探索的节点少了 33%，并且它的执行速度是$M_{max} = +\infty$的两倍，但是 RDms 只有 3%左右。

6 结论

本文研究了在 Petri 网框架下的混合调度策略，时延 Petri 网为表示柔性制造系统中经常遇到的并发活动、共享资源和优先约束提供了一种有效的方法，我们使用一种混合启发式算法，在 Petri 网可达性图中

局部执行 A*算法并全局执行 BT 算法，以搜索具有不同批量的简单制造系统的接近最优的调度方案，为了减少拒绝关键标识的可能性，算法中还采用了一种改进的对先前生成标识的检查方法，最后该算法被用于一组具有有限缓冲区大小、可选路径和双重资源的更复杂的柔性制造系统。

我们将设定不同的表现指标如延时最小化来推进后面的工作，还将研究柔性制造系统中启发函数的效率。

致谢

本文所述研究得到了国家教育部博士基金(批准号 20050288015)和 NUST 科技创新基金的资助。作者要感谢朗讯科技的华信亨利雄博士在这项研究中提出的有益建议。

参考文献

1. Tzafestas S, Triantafyllakis A (1993) Deterministic scheduling in computing and manufacturing systems: a survey of models and algorithms. Math Comput Simul 35:397–434

2. Lee DY , Dicesare F (1994) Scheduling FMS using Petri nets and heuristic search. IEEE Trans Robot Autom 10:123–132

3. Tuncel G, Bayhan GM (2007) Applications of Petri nets in production scheduling: a review. Int J Adv Manuf Technol 34(7–8):762–773

4. Y u H, Reyes A, Cang S, Lloyd S (2003) Combined Petri net modelling and AI based heuristic hybrid search for flexible manufacturing systems—part I: Petri net modelling and heuristic search. Comput Ind Eng 44:527–543

5. Russell S, Norvig P (1995) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River

6. Jeng MD, Chen SC (1998) A heuristic search approach using approximate solutions to Petri net state equations for scheduling flexible manufacturing systems. Int J Flex Manuf Syst 10:139–162

7. Yim SJ, Lee DY (1996) Multiple objective scheduling for flexible manufacturing systems using Petri nets and heuristic search. In: Proceeding of the IEEE International Conference on Systems, Man and Cybernetics: Information Intelligence and Systems, Beijing, China, pp 2984–2989

8. Xiong HH, Zhou MC (1998) Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. IEEE Trans Semicond Manuf 11(3):384–393
9. Reyes A, Y u H, Kelleher G, Lloyd S (2002) Integrating Petri nets and hybrid heuristic search for the scheduling of FMS. Comput Ind 47 (1):123–138
10. Inaba A, Fujiwara F, Suzuki T, Okuma S (1998) Timed Petri net- based scheduling for mechanical assembly-integration of planning and scheduling. IEICE Trans Fundam Electron Commun Comput SciE81-A(4):615–625
11. Mejia G (2002) An intelligent agent-based architecture for flexible manufacturing systems having error recovery capability. Ph.D.thesis, University of Lehigh, USA
12. Huang B, Sun Y , Sun YM (2008) Scheduling of flexible manufacturing systems based on Petri nets and hybrid heuristic search. Int J Prod Res 46(16):4553–4565
13. Zeng QL, Wan LR (2007) Modeling and analysis for a kind of flexible manufacturing system involving time factors. Int J Adv Manuf Technol 34(3–4):346–352
14. Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, USA
15. Huang B, Sun YM (2005) Improved methods for scheduling flexible manufacturing systems based on Petri nets and heuristic search. J Contr Theor Appl 2:139–144

Int J Adv Manuf Technol (2010) 48:925–933
DOI 10.1007/s00170-009-2329-8

ORIGINAL ARTICLE

A hybrid heuristic search algorithm for scheduling FMS based on Petri net model

Bo Huang · Yu Sun · Ya-Min Sun · Chun-Xia Zhao

Received: 16 May 2008 / Accepted: 18 September 2009 / Published online: 10 October 2009
© Springer-Verlag London Limited 2009

Abstract This paper presents a new scheduling method for a flexible manufacturing system (FMS) in a Petri net framework. Petri nets can concisely model multiple lot sizes for each job, the strict precedence constraint, multiple kinds of resources, and concurrent activities. To decrease the likelihood of rejecting the critical markings, our algorithm adopts an improved checking method for previous generated marking. To reduce the computation complexity, an elaborate scheme is applied, which performs A* search locally and backtracking search globally in the reachability graph of the Petri net. Some numerical experiments are carried out to demonstrate usefulness of the algorithm.

Keywords Petri nets · Manufacturing strategy · Scheduling · Heuristics

1 Introduction

In a manufacturing system, scheduling is a typical combinatorial optimization problem, which decides starting times and allocations of jobs to be processed. A desirable method must include two characteristics: (1) easy formulation of the problem and (2) quick identification of semioptimal solutions (with small computational efforts). So, many industry and research communities are now focusing on developing methods for solving real-world flexible manufacturing system (FMS) scheduling problems. However, no perfect solution has been found for all problems, due primarily to the complexity of FMS scheduling. The general FMS scheduling problem belongs to one of the NP-hard combinatorial problems [1] for which the development of optimal polynomial algorithm is unlikely.

The performance of FMS has been recently studied by the Petri net community. It is appropriate to select Petri net-based algorithms for optimization purposes. Despite their popularity as a modeling tool, Petri nets have not received much attention for optimization purposes because of the intractability of their state space [2, 3]. Recent approaches have attempted to use artificial intelligence techniques [3, 4] to selectively search the Petri net reachability graph using the well-known A* search algorithm (see Russell and Norvig [5]). The A* search algorithm presented in [2, 4, 6–9] developed for minimizing the objective function of flow time of parts in the system are all derived from the original algorithm presented in [2]. However, the origin algorithm is not admissible in certain conditions, i.e., it does not always guarantee for an optimal solution even with admissible heuristic function. This problem was also mentioned by Yu et al. [4] who devised a remedy for this problem. But the improved method still cannot guarantee not rejecting markings that lead to the optimal solutions [4].

Another problem observed is the difficulty in finding an optimal or near-optimal solution in a reasonable amount of time for a sizable problem. Some improved algorithms have been introduced to reduce the search effort. Xiong and Zhou [8] have studied hybrid algorithms combining best-first (BF) and backtracking (BT)

B. Huang (✉) · Y.-M. Sun · C.-X. Zhao
School of Computer Science and Technology,
Nanjing University of Science and Technology,
Nanjing, People's Republic of China
e-mail: star_njust@yahoo.com.cn

Y. Sun
School of Mechanical Engineering,
Nanjing University of Science and Technology,
Nanjing, People's Republic of China

search methodologies. The A* algorithms which limit the BT capability of the algorithm by introducing irrevocable decisions were implemented in [6, 10, 11]. Some hybrid search algorithms based on the relaxation of the evaluation scope of an A*-based algorithm have been considered in [4, 9].

In this paper, we present a hybrid scheduling strategy based on the timed Petri nets. The search scheme adopts (1) an improved checking method for previous generated marking to decrease the likelihood of rejecting the critical markings and (2) a more elaborate scheme, which performs BF search locally and BT search globally, than that of Xiong and Zhou [8]. Then, the scheduling results of the method are derived and compared with that of algorithm BF-BT and BT-BF through an example with different sets of lot sizes. The algorithm is also applied to a set of randomly generated more complex FMS with such characteristics as buffers with limited sizes, operations with multiple resources, and jobs with alternative routings.

2 Modeling and scheduling of FMS based on Petri net structures

A Petri net is defined as a bipartite directed graph containing places, transitions, and directed arcs connecting places to transitions and transitions to places. Pictorially, places are depicted by circles and transitions as bars or boxes. A place is an input (output) place of a transition if there exists a directed arc connecting this place (transition) to the transition (place). A place can contain tokens pictured by black dots. It may hold either none or a positive number of tokens. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. Thus, a marked Petri net can be used to study the dynamic behavior of the modeled discrete event system.

Formally, a Petri net [12, 13] is defined as $Z=(P, T, I, O, m_0)$ where:

- P $\{p_1, p_2, \ldots, p_n\}$, $n>0$ is a finite set of places;
- T $\{t_1, t_2, \ldots t_s\}$, $s>0$ with $P \cup T = \emptyset$ and $P \cap T = \emptyset$is a finite set of transitions;
- I $P\times T\rightarrow\{0, 1\}$ is an input function or direct arcs from P to T;
- O $P\times T\rightarrow\{0, 1\}$ is an output function or direct arcs from T to P;
- m $P\rightarrow\{0, 1, 2, \ldots\}$ is a $|P|$ dimensional vector with $m(p)$ being the token count of place p, m_0 is an initial marking.

In order to simulate the dynamic behavior of a system, a state or marking in a Petri net is changed according to the following transition (firing) rules:A transition is enabled if $m(p)\geq I(p, t)$ for any $p\in P$.An enabled transition t can fire at marking m', and its firing yields a new marking, $m(p) = m'(p) + O(p,t) - I(p,t)$, for arbitrary p from P.

The marking m is said to be reachable from m'. Given Z and its initial marking m_0, the reachability set is the set of all marking reachable from m_0 through various sequences of transition firings and is denoted by $R(Z, m_0)$. For a marking $m\in R(Z, m_0)$, if no transition is enabled in m, then m is called a deadlock marking, and the corresponding system is in a deadlock state.

In this paper, we assume that all jobs are available at the beginning and no new jobs arrive to the system. For these assumptions, we follow the modeling methodology presented in [2, 8, 11]. A place represents a resource status or an operation, a transition represents either the start or completion of an event or operation process, and the stop transition for one activity will be the same as the start transition for the next activity following [2]. Token(s) in a resource place indicates that the resource is available. A token in an operation place represents that the operation is being executed and no token shows none being performed. A certain time may elapse between the start and the end of an operation. This is represented by associating timing information with the corresponding operation place.

An event-driven schedule is searched in a timed Petri nets framework to achieve minimum or near minimum makespan. This paper employs deterministic timed Petri nets by associating time delays with places. The transitions can be fired with a zero duration which is consistent with the definition of nontimed Petri nets. In the Petri net model of a system, firing an enabled transition changes the token distribution (marking). A sequence of firings results in a sequence of markings and all possible behaviors of the system can be completely tracked by the reachability graph of the net. The search space for the optimal event sequence is the reachability graph of the net, and the problem is to find a firing sequence of the transitions in the Petri net model from the initial marking to the final one. The A* algorithm applied to Petri nets was first introduced by Lee and Dicesare [2]. The A* search is an informed search algorithm that expands only the most promising branches of the reachability graph of a timed place Petri net. The basic algorithm [2] is as follows:

Algorithm 1 (A)*

1 Put the initial marking m_0 on the list OPEN.
2 If OPEN is empty, terminate with failure.
3 Remove the first marking m from OPEN and put m on the list CLOSED.
4 If m is the final marking, construct the optimal path from the initial marking to the final marking and terminate.

5 Find the enabled transitions of the marking m.
6 Generate the next marking, or successor, for each enabled transition, and set pointers from the next markings to m. Compute $g(m')$ for every successor m'.
7 For every successor m' of m:
   a if m' is already on OPEN, direct its pointer along the path yielding the smallest $g(m')$.
   b if m' is already on CLOSED, direct its pointer along the path yielding the smallest $g(m')$. if m' requires pointer redirection, move m' to OPEN.
   c Calculate $h(m')$ and $f(m')$, and put m' on OPEN.
8 Reorder OPEN in the increasing magnitude of f.
9 Go to step 2.

The function $f(m)$ in algorithm 1 is calculated from the following expression: $f(m)=g(m)+h(m)$. $g(m)$ represents the makespan of the partial schedule determined so far. On the other hand, $h(m)$, called the *heuristic function*, represents an estimate of the remaining cost (makespan) to reach the marking that represents the goal state m_f. The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which transition to fire first.

If $h(m)$ is a lower bound to all complete solutions descending from the current marking, i.e.:

$$\mathrm{h(m)} \leq \mathrm{h^*(m)}, \forall \mathrm{m} \tag{1}$$

where $h^*(m)$ is the optimal cost of paths going from the current marking m to the final one, the $h(m)$ is admissible, which guarantees for an optimal solution [14].

At each step of the A* search process, the most promising of the markings generated so far is selected. This is done by applying an appropriate heuristic function to each of them. Then, it expands the chosen marking by firing all enabled transitions under this marking. If one of successor markings is a final marking, the algorithm quits. If not, all those new markings are added to the set of markings generated so far. Again, the most promising marking is selected and the process continues. Once the Petri net model of the system is constructed, given initial and final markings, an optimal schedule can be obtained using the above algorithm with admissible heuristic function [2].

3 Checking for previous generated markings

In the above search process, when a new marking is obtained, as a successor to the node currently explored, it can be compared with all the markings generated up to that point. The same markings may represent different paths to achieve the same state. A condition must be checked to decide whether the new path is more promising than the existing one. In algorithm 1, the simple test of comparing markings and the current makespan does not satisfy this, thus it may make the algorithm reject the path that leads to an optimal solution [15].

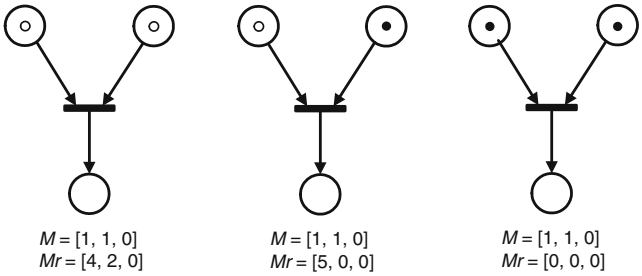

Fig. 1 Transition firing description

This problem was also recognized by Yu et al. [4] who devised a remedy for the problem using the following alternative makespan cost function to replace $g(.)$ when comparing markings:

$$j(M) = g(M) + \sum t_i \Big/ |U_M| \tag{2}$$

where $\sum t_i$ is the sum of the remaining time of the unavailable tokens and $|U_M|$ is the number of the unavailable tokens (an unavailable token in a place means that, although the token is not ready yet, it will become available after a fixed time). However, it still cannot guarantee not rejecting markings that lead to the optimum solutions [4].

To address these issues, we have to understand the process of the timed Petri net state representation. Since the concept of time delay is associated with places, tokens can have two possible states. A token in a place p may mean (1) this token can already be considered as an input token for any transition that has p as an input place or (2) although the token is not ready yet, it will become available after a fixed time (it often happens in the operation places). So, in the second condition, there exists a time left (t_l) for this kind of tokens to be available. As time passes by, t_l is decreased until it reaches zero. When t_l reaches zero, these

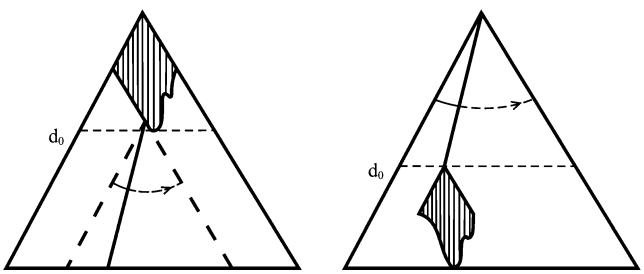

Fig. 2 **a** BF search on top (*shaded area*) followed by BT ending and **b** BT start (*left-to-right arrow*) followed by BF ending

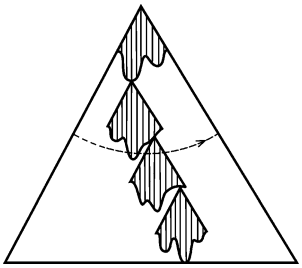

Fig. 3 Perform BF locally and BT globally

tokens become available. Thus, additional time t_l may be required to actually "reach" m. So, the conditions for path updating $m' = m^{\mathrm{O}} \wedge g(m') < g(m^{\mathrm{O}})$ in step 7a (or $m' = m^{\mathrm{C}} \wedge g(m') < g(m^{\mathrm{C}})$ in step 7b) of algorithm 1 and $m' = m^{\mathrm{O}} \wedge j(m') < j(m^{\mathrm{O}})$ in Yu et al. [4] do not necessarily imply a new better path from m_0 to m' (m_0 is the initial marking, m^{O} and m^{C} represent a marking on list OPEN or CLOSED, respectively). In this section, we propose an alternate checking method that considers markings and remaining process time of tokens simultaneously for selecting the better node. To express our method, we need the definition of *timed marking*.

Definition 1: timed marking of a timed Petri net A timed marking m for a timed Petri net is a three-tuple (M, Mr, g) where:

- M is the marking of both available and unavailable tokens in the net;
- Mr is a vector containing remaining process time of tokens in each place;
- g is the cost of path to reach M from M_0.

We use M_m, Mr_{m}, and $g(M_m)$ to represent the elements of a time-marking m. Now let us consider the three states of a timed Petri net where M is the same for these cases (Fig. 1). Then, if $g(M_m)<g(M^{\mathrm{O}})$ and one by one the values of Mr_m are not bigger than that of Mr^{O}, it can be established that a better path between M_0 and M_m was found (the last condition is denoted as $Mr_m \leq Mr^{\mathrm{O}}$ in this paper). This is described in algorithm 2.

Algorithm 2 (improved A)*

1 Place the initial marking m_0 on the list OPEN.
2 If OPEN is empty, terminate with failure.
3 Remove the first marking m from OPEN and put m on the list CLOSED.
4 If m is the final marking m_f, construct the optimal path from m back to m_0 and terminate.
5 Find the set of enabled transitions $\{t_j\}$ (j=1…$et(m)$). $et(m)$ is number of enabled transitions when the marking is m.
6 Generate the children markings m'_j that would result from firing each enabled transition t_j and calculate $g(m'_j)$, $h(m'_j)$, and $f(m'_j)$.
7 For each of the marking M'_j, do the following:
   a If m'_j is equal to some marking m^{O} already on OPEN, compare Mr and g. If $Mr'_j \geq Mr^{\mathrm{O}} \wedge g(m'_j) \geq g(m^{\mathrm{O}})$, go to step 2; If $Mr'_j \leq Mr^{\mathrm{O}} \wedge g(m'_j) < g(m^{\mathrm{O}})$, delete m^{O} from OPEN and insert m'_j on OPEN. Otherwise, insert m'_j on OPEN.
   b If m'_j is equal to some marking m^{C} already on CLOSED, compare Mr and g. If $Mr'_j \geq Mr^{\mathrm{C}} \wedge$

Table 1 Scheduling results of the example for lot size (5, 5, 2, 2)

Algorithm	Depth of BF or $M_{\max}$	Makespan	Number of markings	CPU time (s)
A*		58	3,437	14
BT		105	85	571
BF-BT	20	94	571	0.65
	40	85	1,607	4
	50	79	2,123	6
	60	74	2,775	8
	80	64	3,308	11
BT-BF	20	88	248	0.38
	40	80	484	0.8
	50	70	1,247	3.6
	60	64	1,520	6.5
	80	62	1,687	7
A* locally and BT globally	1	75	230	0.17
	5	68	358	0.31
	10	67	634	0.62
	15	65	540	0.46
	20	62	946	0.78
	25	62	921	0.93
	30	60	1,423	1.12

Table 2 Scheduling results of the example for lot size (8, 8, 4, 4)

Algorithm	Depth of BF or$M_{\max}$	Makespan	Number of markings	CPU time (s)
A*		100	9,438	112
BT		198	145	0.23
BF-BT	40	168	3,888	24
	60	154	5,234	38
	80	140	7,699	49
	100	127	8,819	90
	120	108	9,233	104
BT-BF	40	163	585	1.4
	60	140	1,590	7
	80	121	2,873	18
	100	112	4,545	36
	120	104	8,045	76
A* locally and BT globally	1	128	422	0.31
	5	116	570	0.46
	10	111	782	0.62
	15	112	1,047	1.56
	20	109	1,694	2.65
	25	105	1,585	2.03
	30	105	1,780	2.65

$g(m'_j) \geq g(m^C)$, go to step 2; If $Mr'_j \leq Mr^C \wedge g(m'_j) < g(m^C)$, delete m^C from CLOSED and insert m'_j on OPEN. Otherwise, insert m'_j on OPEN.

c If m'_j is not on either list, insert m'_j on OPEN.

8 Reorder OPEN in the increasing magnitude of f.

9 Go to step 2.

In this algorithm, when a marking is generated we compare its values of M, Mr, and g with that of markings already generated. If it is better than some marking already on OPEN or CLOSED, we insert it on OPEN and delete the marking on OPEN or CLOSED. If worse, we directly reject it. If the quality of the marking cannot be evaluated

Table 3 Scheduling results of the example for lot size (10, 10, 6, 6)

Algorithm	Depth of BF or$M_{\max}$	Makespan	Number of markings	CPU time (s)
A*		134	23,092	720
BT		274	193	0.38
BF-BT	80	206	6,281	64
	100	198	12,341	240
	120	180	16,602	480
	140	169	20,155	540
	160	153	21,797	660
BT-BF	80	209	1,254	5
	100	181	2,315	16
	120	162	8,495	139
	140	150	11,368	390
	160	148	18,875	560
A* locally and BT globally	1	170	581	0.62
	5	155	809	0.93
	10	146	1,089	1.25
	15	147	1,538	2.50
	20	144	1,889	2.81
	25	142	2,850	5.62
	30	141	2,995	7.03

temporarily, we insert it on OPEN. This method presents modifications to the basic algorithm introduced by Lee and DiCesare [2] and it will not reject the promising markings. Additional details can be seen in [15].

4 A*-BT Combinations

A* search is optimal and optimally efficient. But for sizable scheduling problems, it is very difficult to find the optimal solution in a reasonable amount of time and, in many problems, A* spends a large amount of time discriminating among paths whose costs do not vary significantly from each other [14].

To reduce the memory space and computation time required by a pure A* strategy, Xiong and Zhou [8] have proposed two hybrid search methods (BF-BT and BT-BF) which combine the heuristic A* strategy with the BT strategy based on the execution of the Petri net. BF-BT is depicted in Fig. 2a where the A* strategy is applied at the top of the search graph (represented by the shaded area with the irregular frontier) and a BT strategy at the bottom (represented by the left-to-right arrow). As soon as a depth-bound d_0 is reached, the BT search takes over from the best node on OPEN until the entire graph beneath that node is traversed. Another algorithm BT-BF is shown in Fig. 2b. Here, BT is employed at the top of the graph, whereas A* is used at the bottom. BT is applied until a depth-bound d_0 is reached. At this point, instead of backing up, the A* search is started from the node at d_0 until it returns a solution or exists with failure.

Because the performance of the BF search is usually at its best at the bottom of the search graph, BT-BF performs much better than BF-BT [8]. However, algorithm BT-BF presents two major drawbacks: (1) The important decisions with respect to the quality of a schedule may happen at the early stages of the scheduling activity [8]; this increases the likelihood of missing the critical candidates for the BT-BF search which employs BT search instead of BF search at the early stage. (2) The depth of the solution is usually not known a priori, so it is hard for the BT-BF strategy to decide where to trigger the transition from BT to BF is right.

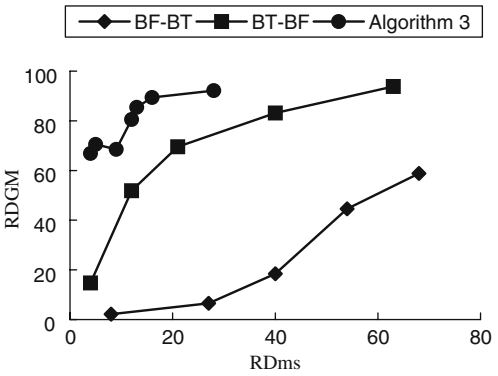

Fig. 5 RDGM versus RDms for lot size (8, 8, 4, 4)

A more elaborate scheme, which performs BF locally and BT globally, is proposed in this section. It is depicted in Fig. 3. We begin searching in an A* manner. At each loop of the search process, we check the number of nodes on OPEN after all successor nodes have been generated from the father node. Once it exceeds a threshold M_{max}, we regard all the nodes on OPEN as direct successors of the root node and submit them to a BT search. BT selects the best among these successors and "expands" it using BF search; that is, it submits the chosen node to a local A* search until the number of nodes on OPEN exceeds M_{max} again and treats the new nodes on OPEN as direct successors of the node "expanded." This strategy is implemented in algorithm 3.

Algorithm 3 (A locally and BT globally)*

1 i=0.
2 Place the initial marking m_0 on the list OPEN$_i$.
3 If i>0, remove the first marking m from list OPEN$_{i-1}$ and put m on list OPEN$_i$
4 If OPEN$_i$ is empty, check i. If i=0, terminate with failure. If i>0, go to step 3.

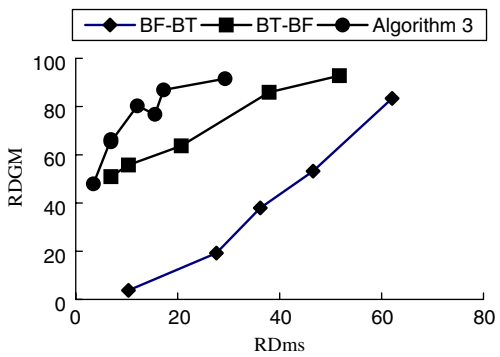

Fig. 4 RDGM versus RDms for lot size (5, 5, 2, 2)

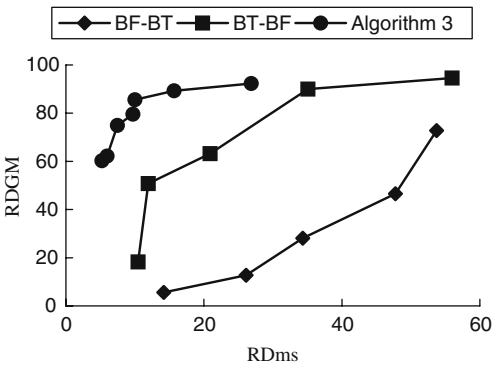

Fig. 6 RDGM versus RDms for lot size (10, 10, 6, 6)

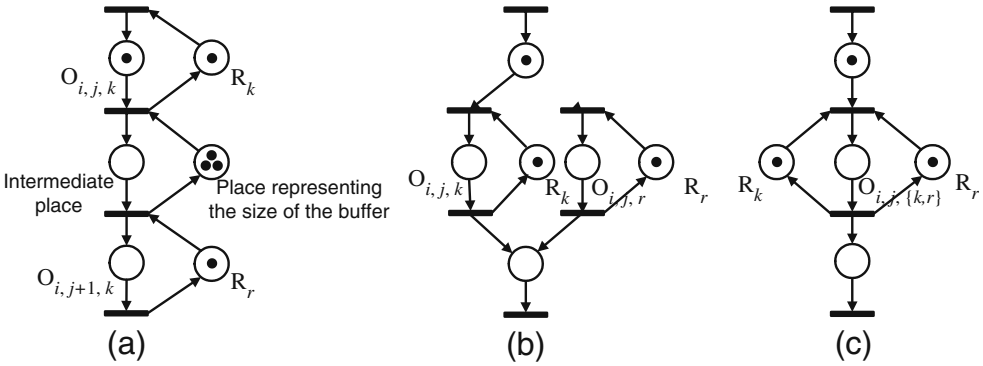

Fig. 7 **a** A model for a buffer with a finite size, **b** a model for a job having alternative routings, and **c** a model for an operation with dual resources

5 Remove the first marking m from list OPEN_i and put m on the list CLOSED_i.
6 If m is the final marking m_f, construct the path from m back to m_0 and terminate.
7 Find the set of enabled transitions $\{t_j\}$ (j=1…$et(m)$). $et(m)$ is number of enabled transitions when the marking is m.
8 Generate the children markings m'_j that would result from firing each enabled transition t_j and calculate $g(m'_j)$, $h(m'_j)$, and $f(m'_j)$.
9 For each of the marking M'_j, do the following:
   a If m'_j is equal to some marking m^{O} already on OPEN_i, compare Mr and g. If $Mr'_j \geq Mr^{\text{O}} \wedge g(m'_j) \geq g(m^{\text{O}})$, go to step 4; If $Mr'_j \leq Mr^{\text{O}} g(m'_j) < g(m^{\text{O}})$, delete m^{O} from OPEN_i and insert m'_j on OPEN_i. Otherwise, insert m'_j on OPEN_i.
   b If m'_j is equal to some marking m^{C} already on CLOSED_i, compare Mr and g. If $Mr'_j \geq Mr^{\text{C}} \wedge g(m'_j) \geq g(m^{\text{C}})$, go to step 4; If $Mr'_j \leq Mr^{\text{C}} \wedge g(m'_j) < g(m^{\text{C}})$, delete m^{C} from CLOSED_i and insert m'_j on OPEN_i; Otherwise, insert m'_j on OPEN_i.
   c If m'_j is not on either list, insert m'_j on OPEN_i.
10 Reorder OPEN_i in the increasing magnitude of f.
11 If the number of nodes on OPEN_i exceeds $M_{\max}$, i=i+1 and go to step 3; otherwise, go to step 4.

In summary, this strategy amounts to running an informed depth-first search where each node expansion is accomplished by a memory-limited A* search and the nodes on OPEN are defined as children.

For the example in Xiong and Zhou [8], the three sets of lot size (5, 5, 2, 2), (8, 8, 4, 4), and (10, 10, 6, 6) are tested. We employ algorithm 3 with the same admissible heuristic function $h(m)$ as that in Xiong and Zhou [8]. The code was written in C# in its entirety. The tests were performed on personal computers having an AMD Athlon microprocessor at a speed of 1 GHz with 512 MB of memory. The scheduling results of makespan, number of generated markings, and computation time are shown in Tables 1, 2, and 3 for lot sizes (5, 5, 2, 2), (8, 8, 4, 4), and (10, 10, 6, 6), respectively. The results for different cases obtained from A* search, BT method, and BF-BT and BT-BF algorithms in Xiong and Zhou [8] are also shown in these tables.

All the algorithms BF-BT, BT-BF, and algorithm 3 cut down the computation complexity by narrowing the evaluation scope at the expense of losing the optimality. The relations of the number of generated markings reduced versus optimality lost are shown in Figs. 4, 5, and 6 for three different sets of lot size (5, 5, 2, 2), (8, 8, 4, 4), and (10, 10, 6, 6) respectively. In these figures, the percentage of optimality lost, which is the comparison of the makespan, is equal to:

$$\text{RDms} = \frac{\text{ms(Hybrid)} - \text{ms(A*)}}{\text{ms(A*)}} \times 100\% \tag{3}$$

and the percentage of computation complexity reduced, which is the comparison of the storage (number of generated markings), is equal to:

$$\text{RDGM} = \frac{\text{GM}\left(\text{A}^*\right) - \text{GM(Hybrid)}}{\text{GM}\left(\text{A}^*\right)} \times 100\%. \tag{4}$$

From the testing results, the following conclusions are drawn.

The hybrid heuristic search which employs A* locally and BT globally in the Petri net reachability graph

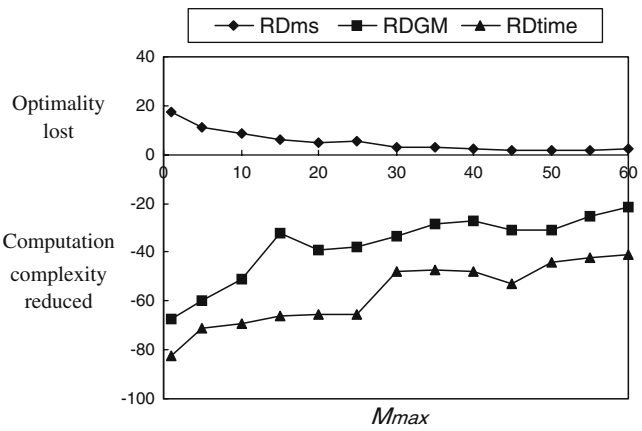

Fig. 8 Performance of algorithm 3

(algorithm 3) performs much better than algorithms BF-BT and BT-BF. This is due to three reasons. Firstly, the performance of heuristic BF search is at its best when its guiding heuristic is more informed, and this usually happens at the bottom of the search graph [14]. Thus, algorithm 3 which employs the A* search (BF search) locally in all stages of the search process greatly reduces the computation complexity compared with the BF-BT search which only employs the A* search at the top of the search graph. Secondly, the important decisions with respect to the quality of a schedule may happen at the early stages of the scheduling activity [8]. So algorithm 3 can more greatly decrease the likelihood of missing the critical candidates than the BT-BF search that employs the BT search instead of the A* search at the early stage. Thirdly, algorithm 3 adopts an improved checking method for previous generated markings and it further decreases the likelihood of rejecting the critical markings.When $M_{\max}=1$, algorithm 3 not only will not degenerate to the BT method, but can still outperform the BT method. This is due to the fact that the two algorithms adopt different policies to node generation. The BT method adopts the last-in-first-out policy to node generation. When a marking is first selected for expansion, only one of its enabled transitions is chosen to fire and thus only one of its successor markings is generated. This newly generated marking is again submitted for expansion. However, in algorithm 3 with $M_{\max}=1$, after a marking has been expanded, the best marking of list OPEN is chosen for the next expansion. Thus, the quality of markings (even the final marking represents the solution found) expanded in each step of algorithm 3 with $M_{\max}=1$ is better than that of the BT method.

5 Application to more complex cases

In this section, we test algorithm 3 with more complex problems which have such characteristics as (1) buffers with limited sizes, (2) jobs with alternative routings, and (3) operations with multiple resources. These characteristics are illustrated in Fig. 7 where $O_{i,\,j,\,k}$ represents the *j*th operation of the *i*th job type being processed with the *k*th resource. When the buffer size is finite, the model can be modified as shown in Fig. 7a. The buffer size is represented by the number of tokens, e.g., the buffer size of the model shown in Fig. 7a is three. In Fig. 7b, the *j*th operation of job *i* can be performed by alternative routings, i.e., by using either resource *k* or resource *r*. In Fig. 7c, the performance of the operation needs multiple resources, resource *k* and resource *r*.

We generated a set of 40 random problems to test the algorithm. These problems, which were generated by the method of randomly selecting and linking predefined Petri net modules [11], had the following characteristics. The system had three resources and four different jobs with three operations each. Seventy-five percent of jobs had two alternative routings and 40% of operations had dual resources. Each operation was assigned a random cost from a uniform distribution (one to 100). The size of each buffer was limited to one to three and the lot size of each job was one to three, too.

We solved the same problem set using algorithm 3 with values of $M_{\max}$ of [1, 5, 10, 15…60]. And we compare these with the results obtained when $M_{\max}$ is set to infinity (pure A* strategy). The scheduling results are summarized in Fig. 8. The uppermost curve represents the mean percentage of optimality lost (RDms), the middle curve represents the mean percentage of search effort reduced (RDGM), and the lowest curve represents the mean percentage of computational time reduced (RDtime, which is the comparison of computational time and is equal to RDtime $= \frac{\mathrm{A*-Hybrid}}{\mathrm{A*}} \times 100\%$). We can see that algorithm 3 can greatly reduce the computational complexity (number of generated markings and computational time), and the mean optimality lost is substantially low, e.g., for $M_{\max}=30$, by average, it explores 33% of nodes less and it executes two times faster than for $M_{\max}=+\infty$, but the RDms is only around 3%.

6 Conclusions

This paper investigates a hybrid scheduling strategy in a Petri net framework. Timed Petri nets provide an efficient method for representing concurrent activities, shared resources, and precedence constraints encountered frequently in FMS. We use a hybrid heuristic algorithm that performs A* locally and BT globally in the Petri net reachability graph to search for a near-optimal schedule of a simple manufacturing system with different sets of lot sizes. In order to decrease the likelihood of rejecting the critical markings, an improved checking method for previous generated marking is also applied in the algorithm. Finally, the algorithm is used for a set of more complex FMS with limited buffer sizes, alternative routings, and dual resources.

Further work will be conducted in setting different performance indices such as minimization of tardiness. The efficiency of the heuristic functions in FMS will also be investigated.

Acknowledgements The research described in this paper is supported by a grant from the doctoral fund of the Ministry of Education of China (Grant No. 20050288015) and the science innovation fund of NUST. The authors want to thank Dr. Huanxin Henry Xiong at Lucent Technologies for his helpful suggestions in this research.

References

1. Tzafestas S, Triantafyllakis A (1993) Deterministic scheduling in computing and manufacturing systems: a survey of models and algorithms. Math Comput Simul 35:397–434
2. Lee DY, Dicesare F (1994) Scheduling FMS using Petri nets and heuristic search. IEEE Trans Robot Autom 10:123–132
3. Tuncel G, Bayhan GM (2007) Applications of Petri nets in production scheduling: a review. Int J Adv Manuf Technol 34(7–8):762–773
4. Yu H, Reyes A, Cang S, Lloyd S (2003) Combined Petri net modelling and AI based heuristic hybrid search for flexible manufacturing systems—part I: Petri net modelling and heuristic search. Comput Ind Eng 44:527–543
5. Russell S, Norvig P (1995) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River
6. Jeng MD, Chen SC (1998) A heuristic search approach using approximate solutions to Petri net state equations for scheduling flexible manufacturing systems. Int J Flex Manuf Syst 10:139–162
7. Yim SJ, Lee DY (1996) Multiple objective scheduling for flexible manufacturing systems using Petri nets and heuristic search. In: Proceeding of the IEEE International Conference on Systems, Man and Cybernetics: Information Intelligence and Systems, Beijing, China, pp 2984–2989
8. Xiong HH, Zhou MC (1998) Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. IEEE Trans Semicond Manuf 11(3):384–393
9. Reyes A, Yu H, Kelleher G, Lloyd S (2002) Integrating Petri nets and hybrid heuristic search for the scheduling of FMS. Comput Ind 47 (1):123–138
10. Inaba A, Fujiwara F, Suzuki T, Okuma S (1998) Timed Petri net-based scheduling for mechanical assembly-integration of planning and scheduling. IEICE Trans Fundam Electron Commun Comput Sci E81-A(4):615–625
11. Mejia G (2002) An intelligent agent-based architecture for flexible manufacturing systems having error recovery capability. Ph.D. thesis, University of Lehigh, USA
12. Huang B, Sun Y, Sun YM (2008) Scheduling of flexible manufacturing systems based on Petri nets and hybrid heuristic search. Int J Prod Res 46(16):4553–4565
13. Zeng QL, Wan LR (2007) Modeling and analysis for a kind of flexible manufacturing system involving time factors. Int J Adv Manuf Technol 34(3–4):346–352
14. Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, USA
15. Huang B, Sun YM (2005) Improved methods for scheduling flexible manufacturing systems based on Petri nets and heuristic search. J Contr Theor Appl 2:139–144