arXiv:2601.14436v1 [cs.NE] 20 Jan 2026

蚁群优化的完整进程代数表示*

María García[1], Natalia López[2], and Ismael Rodríguez[2,3] †‡§¶

摘要

我们提出一种进程代数 PA^{2}CO，可对并行化的蚁群优化算法进行完整、详细的规范描述。在阐释三种不同蚁群优化算法（蚁群系统、 *MAX-MIN* 蚁群系统和蚁群系统）的基础后，我们正式定义 PA^{2}CO，并用它表示采用不同并行方案的多种实现。具体而言，我们对细粒度和粗粒度规范进行了形式化；这两类规范分别利用系统不同粒度层级上的并行执行。

关键词： 蚁群优化，群体智能，进程代数，形式化方法。

1 引言

优化问题在科学、工程学和社会科学领域无处不在。这些问题通常是 NP 难的，这意味着在

*本文发表于 Information Sciences。当前版本为作者接受稿。研究部分得到项目 PID2019-108528RB-C22 的资助，并获得马德里自治区 S2018/TCS-4339（BLOQUES-CM）计划的资助，该计划由欧盟 EIE 基金共同资助。

†1马德里康普顿斯大学数学科学学院，西班牙马德里 28040

‡2 信息系统与计算系，信息学学院。马德里康普顿斯大学。西班牙马德里 28040。§3知识技术研究所。

¶电子邮箱： mgarci51@ucm.es, natalia@sip.ucm.es, isrodrig@sip.ucm.es

标准假设（P $\neq$ NP）下，没有任何算法能在合理的执行时间内（尤其是多项式时间内）求解这些问题。因此，通常会使用次优启发式算法。进化算法和群体智能算法通过反复让一组简单实体依照简单规则彼此交互来寻找次优解，使算法考虑的候选解随着时间推移不断改进。例如，在遗传算法（GA）（Goldberg（1989））和粒子群优化（PSO）（Kennedy、Eberhart 和 Shi（2001）；Poli、Kennedy 和 Blackwell（2007））中，这些候选解本身就是实体（分别为染色体和粒子），它们反复相互影响，使新解趋于与先前的优质解更加相似。另一方面，在蚁群优化（ACO）（Dorigo 和 St¨utzle（2003年））和河流形成动力学（RFD）（Rabanal、Rodr´ıguez 和 Rubio（2007年））中，这些实体（分别为蚂蚁和水滴）会协同在一块由它们共同遍历的画布上绘制出一个解。具体而言，在蚁群优化中，蚂蚁在图上行进，同时沿途留下信息素轨迹；后续蚂蚁则更有可能沿着信息素轨迹较浓的路径前进。通过这种方式，图中最终会形成信息素轨迹显著更浓的路径，而这些路径（例如构成环、树等）就是所考虑问题的解（例如旅行商问题、斯坦纳树问题等）。

需要注意的是，由于环境中实体具有半独立性，一般而言，群体算法和进化算法，尤其是蚁群优化，都可以高效地并行实现（例如参见 Gao、Sun、Zhang、Chen、Gao 和 Zhang（2019年）；Borisenko 和 Gorlatch（2019年）；de Melo Menezes、Herrmann、Kuchen 和 Buarque de Lima Neto（2021）；Le、Nguyen、Garg、Huynh、Bao 和 Tuan（2021）；de Melo Menezes、Kuchen 和 de Lima Neto（2022））。只有当每个进程在其执行时间的相当大一部分内都能执行独立任务（即不需要与其他进程通信的任务）时，并行算法才是高效的。蚁群优化符合这一方案：每个进程都可以在一段时间内处理一只蚂蚁（或一组蚂蚁），仿佛其他进程并不存在；而信息素轨迹可以每经过n 次本地进程迭代后才进行一次全局同步。 n较大时，可以实现较高程度的本地独立性，从而提高执行效率——代价是更长时间地使用过时的轨迹数据，而这反过来可能会妨碍算法收敛到良好解。

进程代数是形式语言，可用于系统地分析并发系统（Milner、Parrow 和 Walker（1992）；N´u˜nezand Rodr´ıguez

（2001）；L´opez 和 N´u˜nez（2004）；Fokkink（2013年））。在这些语言中，系统中按顺序执行的部分以及并行执行的部分（以及它们与哪些其他部分并行）都会得到明确规定，同时也规定了后者何时需要同步。该语言的形式语义使我们能够系统地研究并发事件依照规范可能发生的所有 方式。通常，只需以任意方式反复应用系统规范的操作语义，就能直接推导出一个标记迁移系统，它表示系统所有可能执行构成的树。我们可以利用这个迁移系统，系统地搜索期望或不期望的性质（例如，没有死锁、某些状态必然或可能可达、不发生饥饿的情况下终止等）。

文献中已经提出了一些形式化框架，用于定义一般的进化算法和群智能算法，其中有些还提到了如何将这一模型具体化为蚁群优化。在某些情况下，这些模型并未明确表示系统的并发行为，因此无论执行是顺序的还是并行的，模型都是相同的（Loscos、Mart´ı-Oliet 和 Rodr´ıguez（2022））。在包含并发显式表示的模型中（基于进程代数），抽象层次通常很高，因此蚁群优化行为的底层细节（例如如何进行概率选择、如何更新信息素轨迹等）完全没有得到表示（例如 Eberbach（2007））。已有研究提出使用基于广义网的并发模型来表示蚁群优化（Fidanova 和 Atanassov（2008、2009、2010）），不过，算法变量及其演化过程，以及取决于这些变量的条件，采用的是文本形式而非形式化形式。在这些研究中，引入并行化是为了使算法变量能够在相互独立的操作中并发演化。然而，由于模型通过单个变量对所有蚂蚁的每个特征状态进行抽象，因此并未明确将不同蚂蚁或蚂蚁群并行化为相互分离的进程；而这正是蚁群优化的细粒度和粗粒度并行实现所分别采用的标准方法。此外，还设计了一些蚁群优化进程代数模型，用于建模自然界真实蚂蚁的行为，而不是蚁群优化算法本身（例如 Tofts（1992）；Sumpter、Blanchard 和 Broomhead（2001）；Massink 和 Latella（2012））。

据我们所知，文献中尚未给出任何形式化并发模型（包括进程代数），能够对蚁群优化算法方案及其通常的并行行为作出完整的形式化定义。本文的目标是

开发一种适用于表示蚁群优化（也可能适用于其他群体智能方法）的进程代数语言，并用它对蚁群优化进行建模，全面细致地描述其功能行为（即其动作如何依赖算法自身先前构造的数据）和并发行为（即并行动作如何独立发展并相互通信）。本文考察三种不同的蚁群优化算法：蚁群系统（Dorigo、Birattari 和 St¨utzle（2006年））、最大—最小 蚁群系统（St¨utzle 和 Hoos（2000）；Dorigo 和 St¨utzle（2003年）），以及蚁群系统（Dorigo、Maniezzo 和 Colorni（1996年））。我们将用所提出的语言为这三种算法定义详尽的规格，并分别考察多个并行粒度级别。在每种情况下，将操作语义应用于这些模型，都会得到相应算法的行为，且不作任何抽象。事实上，我们为多种蚁群优化版本建模而提出的进程代数，旨在精准纳入表示目标系统所必需的元素，不多不少。[1] 因此，其设计构成了面向系统的语言构造实例。就用该语言构建的蚁群优化各版本模型而言，据我们所知，这些模型首次对蚁群优化给出了完整详尽的定义，完整呈现了其顺序行为和并行行为。实际上，为蚁群优化的多种并行实现提供详尽的形式化定义，有助于这些方法的标准化：只需引用本文定义的任一种并行方案，就能明确指出所采用并行方案的确切含义，从而避免非正式文字说明有时造成的混淆。此外，我们希望这一完整定义能够成为对其行为进行形式化系统分析的基础，并作为全面形式化其他群体智能算法和进化算法的起点。

值得注意的是，与我们的方法相反，进程代数规格通常会对所描述的系统进行高度抽象。当它们用于表示并发协议标准，或描述并发系统在设计初期或

[1]不足为奇的是，用于表示蚁群优化并发执行的基本语言构造，自然也适用于其他群体智能算法。这包括结合概率与非确定性行为的机制、进程间交换信息的方式，以及概率对这些信息的依赖关系等。然而，在我们的语言中明确研究其他群体智能算法的表示方式，不属于本文的讨论范围。

中期阶段所期望的行为时，这种做法很有用。此外，对复杂系统的抽象描述可以比对系统本身的模型更高效地进行自动化分析。例如，如果忽略（抽象掉）某些或全部变量的内部值及其对条件的影响，那么所得模型可能具有的内部状态集合就会大幅缩小。在这种情况下，通过所有可用的选择，探索系统演化至某个未来时点的所有可能路径，成本会低得多——代价是，模型将以非确定性的方式表示一些原本是确定性的选择，因为这些选择所依赖的变量已被忽略。我们完整定义的规格提供了一个出发点，使人们能够轻松地启用各类所需的抽象。通过忽略控制系统某些行为方面的变量（例如，控制每次更新前必须完成的蚂蚁数量、记录此前经过的节点，甚至表示信息素轨迹值的变量），我们可以得到原始系统的不同抽象——与原始的完整规格相比，它们都更容易对截至某个时点的所有可能执行进行系统研究，但每一种抽象都会遗漏某一方面的行为细节。显然，让详细模型变得更抽象，要比为抽象模型添加细节容易得多，因此，我们的规格是一个合适的起始模型，可据此得到针对具体情况所需的任何定制抽象。

本文其余部分安排如下。第 2 节研究蚁群优化算法的运作方式及其动机，并以旅行商问题作为应用示例，说明我们将在后文建模的三种变体（蚁群系统、最大-最小 蚁群系统和蚁群系统）。此外，本节还研究文献中蚁群优化算法的主要并行化策略：细粒度并行化和粗粒度并行化。第 3 节定义用于 ACO 的进程代数 $\mathrm{PA}^2\mathrm{CO}$；该代数专门用于详细表示蚁群优化算法。随后，我们将在第 4 节和第 5 节分别对上述蚁群优化算法的细粒度和粗粒度并行版本进行形式化。在第 6 节中，我们将比较本文给出的所有规格。最后，第 7 节总结本文的主要结论，并提出未来的研究方向。

2 蚁群优化算法

群体智能是一种成熟的启发式方法，可用于解决难度较高的优化问题，其灵感主要源自动物群体互动（例如鸟类或昆虫）。尤其是蚂蚁的行为启发了大量算法，这些算法统称为蚁群优化（Dorigo 等人（1996年））。所有这些算法都基于同一原理：蚂蚁如何在觅食过程中相互交流。

蚂蚁寻找食物的方式非常高效。搜索开始时，蚂蚁会随机移动。一旦有一只蚂蚁找到食物，它就会返回蚁丘，并沿途留下信息素轨迹。由于蚂蚁会被这些轨迹吸引，后续蚂蚁沿着这条路径前进的概率便会增加，从而使路径上的信息素量增加。较短的路径单位时间内被经过的次数多于较长的路径，而且信息素轨迹会随时间消失，因此最终通往食物的最短路径会留下最强的信息素轨迹。这样，所有蚂蚁最终都会沿着同一条路径前进，也就是蚁丘与食物之间最短的路径。

蚁群优化算法依靠这一过程来优化不同类型的问题。本节将研究其中的三种算法：蚁群系统（Dorigo 等人（2006年））、最大-最小 蚁群系统（St¨utzle 和 Hoos（2000）；Dorigo 和 St¨utzle（2003年）），以及蚁群系统（Dorigo 等人（1996年））。我们将通过一个典型的应用示例来说明这些算法：旅行商问题（TSP）。该问题要求遍历 n个城市，每个城市恰好一次，同时使城市间移动所产生的成本总和最小化。更正式地说，该问题是在由 n个顶点组成的完全加权图中寻找成本最低的哈密顿路径——其中，顶点代表城市，边代表城市之间的连接，其对应的权重则代表相应的旅行成本。在本文余下部分，我们将使用表1中的变量来表示这些组成部分，以及每条边在各个时刻的信息素量。

v_i =对应城市的顶点 i	
$e_{i,\ j}$ = 连接这些顶点的边 v_i 和v_j	$\forall\ i,\ \ j \in \{1, \cdots, n\}$和 $i \neq j$
$d_{i,\ j}$ = 边的权重 $e_{i,\ j}$	
$\tau_{i,\ j}(t) = e_i$上的信息素量，$_j$ 在时间 t	

表1：TSP 的变量集合。

2.1 蚁群系统

受上述蚂蚁行为启发的第一个算法由 Marco Dorigo 于 20 世纪 90 年代在其博士论文中提出，称为蚁群系统（AS）（Dorigo 等人（2006年））。如上所述，我们将使用旅行商问题（TSP）来说明该算法的运行机制。

在蚁群系统算法中，有若干只蚂蚁遍历图并生成哈密顿路径。与自然界中的蚂蚁不同，这些蚂蚁具有记忆能力，并生活在一个时间离散的环境中。记忆有助于它们记住已经访问过的顶点（从而避免再次访问）；由于时间是离散的，在时刻 t 作出的决策会在时刻 $t+1$ 执行。每只蚂蚁代表一个解（若尚未经过所有顶点，则为部分解；若已完成一整圈，则为完整解）。

算法会将一定数量的蚂蚁（在我们的例子中为 m 只蚂蚁， m 是预先确定的自然数）分配到图的各个顶点上，使每只蚂蚁都根据某些概率决定接下来访问哪个顶点。这些概率取决于信息素轨迹的强度和每条边的成本。因此，在选择其他蚂蚁最常使用的边（这会形成一种自我诱发的反馈过程）与选择成本最低的边（这本身隐含了一种简单的贪心策略）之间，存在一定的平衡。此外，由于每只蚂蚁都会记住自己已经访问过的顶点，因此可以将选择通向重复顶点的边的概率设为零，以满足问题的限制。具体而言，蚁群系统中提出的第 k 只蚂蚁在时刻选择边 $e_{i,\ j}\ \ t$ 的概率公式为：

$$p_{i,j}^{k}(t) = \begin{cases} \frac{[\tau_{i,j}(t)]^{\alpha} \cdot [\eta_{i,j}]^{\beta}}{\Sigma_{r \in allowed_k} [\tau_{i,r}(t)]^{\alpha} \cdot [\eta_{i,r}]^{\beta}} & \text{if } v_j \notin path_k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

其中：

- $\eta_{i,\ j} = \frac{1}{d_{i}}{}_{.\ j}$ 是边 $e_{i,\ j}$的可见度，
- α， β 是用于控制信息素轨迹与可见度相对重要性的参数，且
- 路径$_k$ 是第 k 只蚂蚁已访问过的顶点列表。

蚂蚁在时刻 t 选择要移动到的顶点后，会在瞬间 $t+1$ 执行移动。到达该顶点后，它会再次进行概率选择。在每个时刻，每只蚂蚁都会做出决策并移动，因此在时刻

$t+n$ 所有蚂蚁都将完成一圈（或一条哈密顿路径）。此时，蚁群系统算法会根据以下公式更新每条边 $e_{i,j}$ 的信息素轨迹：

$$\tau_{i,j}(t+n) = (1-\rho) \cdot \tau_{i,j}(t) + \Sigma_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{2}$$

其中：

- ρ is the pheromone decay coefficient in the time interval $(t, t+n)$,
- $\Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if the k-th ant uses the edge } e_{i,j} \text{ on its lap} \\ 0 & \text{otherwise} \end{cases}$
- Q is a constant, and
- L_k is the total length of the lap of the k-th ant.

这些值 $\tau_{i,j}(0)$ 必须在算法开始时初始化，通常将其设为很低的值。直观地说，该公式表示：蚂蚁完成一圈后，边$e_{i,j}$ 上的信息素轨迹等于当时尚未蒸发的信息素量，加上所有经过该边的蚂蚁在其路径上留下的信息素量。

最后，达到最大迭代次数或所有蚂蚁都走出相同路径时，算法将结束。

2.2 *MAX-MIN* 蚁群系统

MAX-MIN 蚁群系统（简称 $\mathcal{MM}$AS）（St¨utzle 和 Hoos（2000）；Dorigo 和 St¨utzle（2003年））与蚁群系统的区别在于信息素轨迹的更新方式。在蚁群系统中，每只蚂蚁都会根据其一圈中经过的边更新各边上的信息素量；而在 $\mathcal{MM}$AS 中，只有路径成本最低的蚂蚁会更新信息素轨迹，这引入了一种精英策略。此外，在这一版本中，信息素轨迹的取值范围受到限制，以避免搜索算法陷入停滞。因此，信息素轨迹的更新方式如下：

$$\tau_{i,j}(t+n) = \left[(1-\rho) \cdot \tau_{i,j}(t) + \Delta\tau_{i,j}^{best} \right]_{\tau_{min}}^{\tau_{max}} \tag{3}$$

其中：

- $\Delta\tau_{i,j}^{best} = \begin{cases} \frac{1}{L_{best}} & \text{if } e_{i,j} \in edges_{best} \\ 0 & \text{otherwise} \end{cases}$

  being $edges_{best}$ the list of edges chosen by the ant with the best path,

- L_{best} is the length of the path with minimum cost,

- τ_{min}, τ_{max} is the minimum and maximum trail quantity values,

且算子 $\left[\ \right]_b^a$ 定义如下:

- $\left[x\right]_b^a = \begin{cases} a & \text{if } x > a \\ x & \text{if } b \leq x \leq a \\ b & \text{if } x < b \end{cases}$

2.3 蚁群系统

对于蚁群系统 ACS（Dorigo 等人（1996年）），其最大的贡献是在 AS 中每完成一圈后进行的信息素更新之外，引入了局部信息素更新；我们现在将前者称为离线 更新。因此，每只蚂蚁都会以概率方式选择一条边$e_{i,\ j}$，并在选择后立即按照以下公式更新信息素轨迹 $\tau_{i,\ j}$：

$$\tau_{i,j}(t+1) = (1-\rho) \cdot \tau_{i,j}(t) + \rho \cdot \tau_{i,j}(0) \tag{4}$$

其中 $\rho \in (0,1]$ 是信息素衰减系数。降低蚂蚁经过的边上的信息素轨迹强度，旨在使搜索更加多样化。通过这些局部更新，可以降低两只蚂蚁在同一次迭代中走过相同路径的可能性。此外，离线 更新采用了 $\mathcal{MM}$AS 的精英策略，也就是说，只有找到最优路径（成本最低路径）的蚂蚁才会进行更新，具体公式见（3）。可以采用上一轮迭代的最优路径，也可以采用迄今为止找到的最优路径。

与蚁群系统相比，另一项重要变化是概率的计算。在此版本中，选择一条边的概率还取决于一个在 [0,1] 中均匀分布的变量q 以及参数 q_0。因此，蚂蚁按照所谓的伪随机比例规则选择下一个顶点：选择满足 $j^* = argmax_{j\in[1,n]}\left\{\tau_{i,\ j} \cdot \eta_{i\ ,\ j}^{\beta}\right\}$ 的顶点 v_{j^*} 的概率（也就是说，选择信息素数量与可见度的乘积最大、且可见度相对重要性为 β的顶点）为 q_0；而

按照公式 (1) 的标准概率进行选择的概率为$1 - q_0$。

这些信息素更新和概率赋值方式上的变化，相较于搜索空间的探索，提高了先前蚂蚁所收集信息的重要性。

2.4 蚁群优化算法的并行化

下面讨论文献中最常见的这类算法并行化方法。通常，采用 n 个处理器的并行系统，效率不如速度快 n 倍的单个处理器。不过，并行系统通常成本更低，因此人们倾向于将原本顺序执行的进程并行化。并行化蚁群优化算法等元启发式算法，主要是为了在给定时间内提高所得解的质量，或者在要求解质量达到一定标准的情况下，用更短的时间找到足够好的解。

在蚁群优化算法的并行化方面，文献中主要有两种策略：细粒度 并行化和粗粒度 并行化（Dorigo 等人（2006））。细粒度 并行化是指为每个处理器分配很少的个体，使它们之间频繁通信。粗粒度 并行化则是指为每个处理器分配较大的子种群，并减少通信。

已有实证证明（Manfrin、Birattari、St¨utzle 和 Dorigo（2006）），粗粒度 是并行实现蚁群优化算法最高效的策略。事实上，效果最好的版本是：以不同的变量初始化方式，独立执行 n 个相同的顺序蚁群系统算法副本。在这种实现中，会设定最大迭代次数，各算法副本之间不进行任何通信，并从不同副本得到的结果中选出最优解。这样一来，典型蚁群优化并行实现中所有迭代结束时进行的高开销通信，就被替换为所有蚁群优化实例执行结束时的单一通信阶段。

（de Melo Menezes 等人（2022））综述了群体智能算法的并行化研究。研究工作根据其并行化方案和所用硬件类型进行分类。对于蚁群优化，该论文考虑了三种并行化类型：

- 朴素并行化：这种方法为每只蚂蚁分配一个线程。这被认为是一种相当直接的方法，能够实现显著的加速效果，尤其是在优化问题的维度增大时。

- 细粒度并行化：如果核心数量多于蚂蚁数量，则可以为每只蚂蚁分配一个线程块，并利用该线程块中的线程并行执行蚂蚁的内部计算。信息素沉积可以并行化，但冲突的更新必须使用原子操作按顺序计算。总体而言，细粒度 并行化可以缩短执行时间，但无法在硬件层面实现高效的同步方法。

- 并行蚁群：通过为每个蚁群分配一个进程，实现多蚁群并行处理。由于蚁群优化是一种精英算法，只有成功的蚂蚁及其路径会影响蚁群在下一次迭代中的行为，因此通常会丢弃其他蚂蚁执行的大部分工作，以跟随成功的蚂蚁。通过让多个蚁群同时运行，可以产生更多样化的成功解。由于每个子蚁群独立运行，因此需要在它们之间进行同步，以使信息素沉积水平保持一致，不过建议只偶尔同步，以避免增加额外耗时。

为了涵盖适用于蚁群优化算法的不同并行化类型，本文将构建一种进程代数，使我们能够描述从几乎持续通信的细粒度实现，到完全不交换信息、效率最高的粗粒度实现。结合以往文献，我们的细粒度方案处于此前提到的 de Melo Menezes 等人（2022）提出的前两种并行方案之间：每只蚂蚁都表示为一个进程，如所谓的朴素方案；而信息素轨迹值的维护职责则封装在一个额外的进程中，我们称之为图。这样，某些操作便会在蚂蚁进程之外执行——这在某种程度上与该文所称的细粒度方案相似。[2] 此外，我们的粗粒度

[2]该图进程的作用是以简单的方式实现图的更新过程，但显然也可以采用其他表示方式（例如，所有蚂蚁都可以远程访问一个表示图状态的共享变量，或者将图状态分布到所有蚂蚁中，等等）。

方案将是 Manfrin 等人（2006）所述的效率最高的方案，正如前文提到的：运行多个完全独立的蚁群优化算法实例，最后从中选出所有实例找到的最优解。不过，我们也会考虑该方案的一种变体，其中子群体之间可以偶尔同步，从而使它们共同演化出一个解；这与此前提到的 de Melo Menezes 等人（2022）的第三种方案，即所谓的并行蚁群方案相同。

3 进程代数

进程代数是数学结构，可用代数术语表达系统的行为，从而有助于对系统的正确性及其性质进行形式化推理。每个进程代数都由一组算子和一组原子动作构成，原子动作表示不可分割的行为（Fokkink（2000），Groote 和 Reniers（2002））。基于这些算子和动作，可以为表示进程的代数项构建语法，并通过操作语义定义其演化。

动作分为两类：内部动作和通信动作。内部动作是一种原子动作，其执行无法被外部观察者察觉。它记作 μ.[3] 我们考虑 $Act_\mu = Act \cup \{\mu\}$，其中 Act 是通信动作集（我们假设 $\alpha \in Act_\mu$，以便在需要时将通用动作记作 α）。通信动作与进程间信息的发送和接收有关。在我们的进程代数中，我们将考虑发送动作和接收动作，它们通过同步通信执行。此外，我们还将纳入概率信息，以表示某些动作的执行取决于概率时系统的行为。有关概率代数的更多详细介绍，参见（van Glabbeek、Smolka、Steffen 和 Tofts（1990），L´opez 和 N´u˜nez（2004））。

我们将考虑一种非确定性概率模型，即同时使用概率选择和非确定性选择。在这种情况下，通常会考虑两种不同类型的转移：一种与概率相关，另一种与动作相关。在这类模型中，我们可以区分交替模型和非交替模型。在交替模型中，

[3]在文献中，这种内部动作通常记作 τ。然而，当我们对蚁群优化算法进行形式化时，这种记法会与信息素轨迹（也用该符号表示）产生冲突。

概率选择与非确定性选择之间存在严格的交替，而非交替模型则没有这一限制。

在我们的情形中，我们将构建一个非确定性、非交替的概率模型。在这类模型中，关键在于确定每次转移后我们希望到达哪种状态。

3.1 用于 ACO 的进程代数，PA^{2}CO

在我们的情形中，我们希望定义一种概率进程代数，用于表示蚂蚁与图之间的数据发送，以及对这些数据所做的更改。为此，我们将引入状态和配置的概念。正如我们将看到的，状态表示进程在每个时刻的变量取值，并会随着进程本身的演化而改变。配置是由一个进程和一个状态（即与其变量相关联的状态）构成的有序对。

在给出进程代数的语法之前，我们将先介绍若干概念和一些辅助定义。

定义 1 (状态)。 设 Var 为变量集合，$Data$ 为数据集合。我们将任意偏函数 $\sigma : Var \dashrightarrow Data$ 定义为一个状态。所有状态的集合记为 St。

请注意，全函数用 $\longrightarrow$ 表示，而 $\dashrightarrow$ 将用于表示偏函数。

这些状态将由我们进程代数中的进程通过通信动作发送。接下来，我们将定义一些关于状态变量的函数：

定义 2 (限制于集合的状态). 设 $\sigma \in St$ 是一个状态，$V \subset Var$ 是变量的一个子集。我们将 σ 限制于 V 的状态定义为仅考虑 σ 对 V 中变量的赋值后得到的状态，记作 σ_V。也就是说，

$$\forall x \in Var,\ \sigma_V(x) = \begin{cases} \sigma(x) & if\ x \in V \\ undefined & otherwise \end{cases}$$

另一方面，进程一旦开始演化，状态中的变量赋值就会发生变化。因此，我们将考虑一组用于更新状态的函数 $Trans$ 。

定义 3 (变换**)。** 我们将变换定义为作用于状态的函数，即 $trans: \ St \to St$。我们用 *Trans* 表示给定状态集的所有可能变换所构成的集合。

最有用的变换是那些改变状态中单个变量的变换。

定义 4 (单一变换**)。** 设 $\sigma \in St$, $y \in$ 为变量，且 $d \in$ 为数据。我们将变换 $[y \mapsto d]: St \to St$ 应用于 σ，记作 $\sigma[y \mapsto d]$，定义为一种变换：新状态的行为与 σ 相同，但变量 y除外，其新值为 d。也就是说，

$$\forall x \in Var, \ \sigma[y \mapsto d](x) = \begin{cases} \sigma(x) & if \ x \neq y \\ d & otherwise \end{cases}$$

当替换多个变量（但并非所有变量）的值时，为了清晰起见，我们将其视为若干个此类单一变换的组合。在操作语义的定义中，我们将考虑用一个状态变换另一个状态。

定义 5 (状态变换**)。** 设 $\sigma, \sigma' \in St$ 为状态。将状态变换 σ' 应用于 σ 所得的状态，是通过在 σ 中替换由 σ'给出的赋值而创建的状态，记作 $\sigma[\sigma']$。也就是说，

$$\forall x \in Var, \ \sigma[\sigma'](x) = \begin{cases} \sigma(x) & if \ \sigma'(x) = undefined \\ \sigma'(x) & otherwise \end{cases}$$

除变换外，我们还考虑条件。条件是根据分配给给定状态中变量的数据返回布尔值（即 true 或 false）的函数。

定义 6 (条件**)。** 设 *Bool* 为布尔值的集合。我们将条件定义为任意函数 $cond: St \to Bool$。我们用 *Cond* 表示状态集合上的所有条件所构成的集合。

因此，在进程执行的某一时刻，条件是否为 true 取决于该时刻其变量的值。

定义 7 (动作**)。** 我们将动作集记为 Act_μ，并将其定义为由以下巴科斯-诺尔范式表达式所定义的动作：

$$\alpha ::= \mu | \ trans | \ c!(\sigma) | \ c?(\sigma')$$

其中，σ、$\sigma' \in St$，$trans \in Trans$，且 c 是通信通道。

动作 μ 表示内部动作，$trans$ 表示对状态的任意转换， $c!(\sigma)$ 是输出通信动作，通过通道 c 发送状态 σ ； $c?(\sigma')$ 是输入通信动作，通过通道 c 接收保存在 σ' 中的状态。当发送完整的当前状态时，我们将省略参数，直接写作 $c!$。

定义了状态、变换、条件和动作后，我们给出进程的语法。

$G ::= N \mid PN ::= stop \mid X \mid \alpha; G \mid [cond]G \mid \Sigma_{i\in I} N_i \mid rec\ X.N$ (非确定性进程)$P ::= X \mid \oplus_{i\in I}\,[p_i]N_i \mid rec\ X.P$ (概率过程)，其中$p_i \in [0,1]$ 对所有 $i \in I$ 和 $\Sigma_{i\in I} p_i = 1$ $cond \in CondX$ 是一个进程变量， I 是一个有限索引集$\alpha \in Act_\mu$

表2：进程的语法。

定义8 (进程)。 进程的语法由表*2*中给出的巴科斯-诺尔范式表达式定义，其中区分了两类进程，

- 非确定性进程的集合 $\mathcal{P}_N$ 和
- 概率过程的集合 $\mathcal{P}_P$ ，

并且我们考虑 $N \in \mathcal{P}_N$、 $P \in \mathcal{P}_P$ 和 $G \in \mathcal{P}_N \cup \mathcal{P}_P = \mathcal{P}$。

根据 $N \in \mathcal{P}_N$ 的巴科斯-诺尔范式表达式，非确定性进程为：

- 停止，即什么也不做的进程。尽管所有进程都会以动作停止结束，为了清晰起见，我们将省略它。因此，进程α；停止 将表示为 α，

- 执行动作 α ，然后表现得像进程G一样的进程 α； G ，

- 进程 [条件]G 的行为类似于：若 G 条件 $cond$ 成立，则.....；否则它无法演化（此时它等同于停止 动作），

- 与非确定性过程N相关联的递归进程 $rec\ X.N$，并且

- $\Sigma_{i\in I}N_i$，非确定性进程之间的选择（作为二元算子应用时，记作 N_1+N_2）。

总之，非确定性进程包括无法演化的进程，以及通过执行集合 Act_μ 中的动作而演化的进程。

另一方面，我们将作出概率决策的进程（$\oplus_{i\in I}\ [p_i]N_i$），以及与概率过程相关联的递归进程（$rec\ X.P$），视为概率过程。正如我们将在操作语义中看到的那样，这些进程通过作出概率决策来演化。我们要求在作出概率决策后，必须执行一个非确定性进程。事实上，这正是蚁群优化规范中期望的行为；若不施加这一限制，反而会不必要地增加操作语义的复杂性。

前一定义中所用算子的优先级从高到低依次为：递归算子、条件 [条件], 的求值、顺序组合 ;、非确定性选择 Σ，以及概率选择 $\oplus$。

请注意，递归进程 $rec\ X.N$ 的演化方式与进程 N 相同，只需将 N 中变量 X 的所有出现处替换为进程 $rec\ X.N$。通过一个简短的例子即可轻松理解这一行为：

例3.1。令 $A\in G$ 为进程：

$$A ::= recX.(\alpha; X)$$

其中 $\alpha\in Act_\mu$。如果将递归展开一次，那么进程 A 可以描述为：

$$A ::= \alpha; (recX.(\alpha; X))$$

该进程执行 α 并再次重复 A。变量 X 的每次出现都可以看作进程 A 的一次递归调用，因此 A 是一个循环执行 α 的进程。

请注意，在$\Sigma_{i\in I}N_i$中可以从条件和动作中进行选择。事实上，我们希望在代数中实现这种行为，正如我们在下一个示例中看到的那样：

示例 3.2. 假设我们有一个进程 Q，它从n 不同的通道接收数据，但只关注每个输入通道收到的第一条数据。此外，它还希望在通道 *channelOut* 请求时发送另一条数据。于是，它的行为可以表示如下：

$$Q ::= rec.X(\Sigma_{i=1}^{n}[\textit{firstData}_i]\text{通道}_i?(\sigma);\ X\ + \textit{channelOut}!; X)$$

其中，条件 $\textit{firstData}_i$ 表示通道 i 接收到的是否为第一条数据。

稍后我们将在对蚁群优化算法建模时运用本例的思路；特别是，当我们希望已完成一圈的蚂蚁等待其他蚂蚁完成各自的一圈，以便更新轨迹并继续搜索时，就会用到这一思路。

请注意，选择算子定义的是一般意义上的选择，尽管这种选择可以是确定性的、外部的、非确定性的，或它们的任意组合。我们将这种选择称为非确定性选择，因为当不存在使其符合其他任何情况的附加条件时，它便是非确定性的，因此确实是最一般的情况。

例 *3.3.* 假设 val 是一个变量，进程 $[val\ =\ 1]a! + [val\ =\ 0]b!$ 中的选择是确定性的，因为每种情况下最多只有一个确定性选择可用。进程 $a?(x) + b?(x)$ 中的选择是外部的，因为它将由其他进程决定。进程 $a! + b!$ 中的选择是不确定性的，进程 $[val\ =\ 1]a!\ + [val\ =\ 1]b!$ 中也是如此。最后，进程 $recX.($[空气充足]喊救命！；X+[空气充足]喊我快淹死了！；X+挥动手臂！；X+接收救生衣？(σ')+接收救生圈？$(\sigma'))$ 中则混合了前面几种类型。

3.2 两步并行组合

在描述算子、状态集合 St 以及进程语法之后，我们定义配置。在给出正式定义之前，先介绍这一概念的主要直观含义。配置是一个二元组 $\ll G,\ \sigma \gg$ 其中 $G \in \mathcal{P}$且 $\sigma \in St$，因此直观而言，$\ll G,\ \sigma \gg$ 表示进程 G 将从其变量的状态 σ开始执行。正如我们将看到的，在蚁群优化算法的规范中，多个进程将并行执行，我们希望每个进程只在本地修改自己的变量。因此，每个进程都有自己的状态，且独立于其他进程。此外，我们将避免，

通过操作语义的构造，避免概率过程所提供的选择与和它并行的非确定性过程所提供的选择之间发生竞争（请注意，前者会量化其选择的可能性，而后者不会，因此二者不能直接竞争）。由于并行组合具有非确定性，将这两类进程组合在一起，所得组合既不是纯概率性的，也不是纯非确定性的。

为避免这种歧义，操作语义会将并行执行分为两个步骤:

- 首先，通过非确定性方式决定不同配置中哪一个“赢得竞赛”，也就是说，决定接下来执行哪一个。

- 其次，如果选中的是概率配置，接下来就进行概率选择；如果选中的是非确定性配置，接下来就执行一个动作。

完成这两个步骤后，我们将得到一个新的并行配置，并再次重复选择和执行过程。因此，我们将认为所有形如 $\|_{i\in I}C_i$ 的配置都是非确定性的，它们会通过一个内部动作转移到形如 $\|^j_{i\in I}C_i$ 的配置，以表明 C_j 中的 $j \in I$ 赢得了竞赛。这样一来，配置 $\|^j_{i\in I}C_i$ 的（概率性或非确定性）性质将取决于 C_j 中进程的性质。

定义 9（配置）。 我们使用以下巴科斯-诺尔范式表达式定义配置的语法:

$$C ::= \ll G, \sigma \gg \mid \|_{i\in I}C_i \mid \|^j_{i\in I}C_i$$

其中， $G \in \mathcal{P}$、 $\sigma \in St$、 I 是有限的索引集合，且 $j \in I$。

表 3 给出了配置的操作语义。我们用 α^* 表示与数据发送无关的动作，即 $\alpha^* \in Trans \cup \{\mu\}$。

- 规则 R1 表明，状态发送和内部动作的执行不会改变状态。

- 规则 R2 指出，如果进程接收到一个状态，那么我们将接收到的状态应用于初始状态，并考虑由此产生的状态变换。

$$R1\ \frac{}{\ll \alpha;G,\sigma \gg \xrightarrow{\alpha} \ll G,\sigma \gg}\ \alpha \in \{c!,\mu\} \qquad R2\ \frac{}{\ll c?(\sigma');G,\sigma \gg \xrightarrow{c?(\sigma')} \ll G,\sigma[\sigma'] \gg}$$

$$R3\ \frac{}{\ll trans;G,\sigma \gg \xrightarrow{trans} \ll G,trans(\sigma) \gg}$$

$$R4\ \frac{cond(\sigma),\ \ll N,\sigma \gg \xrightarrow{\alpha^*} \ll N',\sigma' \gg}{\ll [cond]N,\sigma \gg \xrightarrow{\alpha^*} \ll N',\sigma' \gg}\ \alpha^* \in Trans \cup Act_\mu \qquad R4'\ \frac{cond(\sigma)}{\ll [cond]P,\sigma \gg \xrightarrow{\mu} \ll P,\sigma \gg}$$

$$R5\ \frac{\ll N_j,\sigma \gg \xrightarrow{\alpha_j} \ll N'_j,\sigma_j \gg}{\ll \Sigma_{i\in I} N_i,\sigma \gg \xrightarrow{\alpha_j} \ll N'_j,\sigma_j \gg}\ j \in I \qquad R6\ \frac{}{\ll \oplus_{i\in I}[p_i]N_i,\sigma \gg \longrightarrow_{p_j} \ll N_j,\sigma \gg}\ j \in I$$

$$R7\ \frac{\ll G_j,\sigma_j \gg \xrightarrow{\alpha^*} \ll G'_j,\sigma'_j \gg}{\|_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\mu} \|^j_{i\in I} \ll G_i,\sigma_i \gg}\ \alpha^* \in Trans \cup \{\mu\} \qquad R8\ \frac{\ll G_j,\sigma_j \gg \longrightarrow_p \ll G'_j,\sigma'_j \gg}{\|_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\mu} \|^j_{i\in I} \ll G_i,\sigma_i \gg}$$

$$R9\ \frac{\ll G_j,\sigma_j \gg \xrightarrow{c?(\sigma)} \ll G'_j,\sigma'_j \gg \qquad \ll G_k,\sigma_k \gg \xrightarrow{c!} \ll G'_k,\sigma'_k \gg}{\|_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\mu} \|^j_{i\in I} \ll G_i,\sigma_i \gg}\ j,k \in I$$

$$R10\ \frac{\ll G_j,\sigma_j \gg \xrightarrow{c!} \ll G'_j,\sigma'_j \gg \qquad \ll G_k,\sigma_k \gg \xrightarrow{c?(\sigma)} \ll G'_k,\sigma'_k \gg}{\|_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\mu} \|^j_{i\in I} \ll G_i,\sigma_i \gg}\ j,k \in I$$

$$R11\ \frac{\ll G_j,\sigma_j \gg \longrightarrow_p \ll G'_j,\sigma'_j \gg}{\|^j_{i\in I} \ll G_i,\sigma_i \gg \longrightarrow_p \|_{i\in I} \ll G'_i,\sigma'_i \gg}\ \ll G'_i,\sigma'_i \gg = \ll G_i,\sigma_i \gg \forall i \neq j$$

$$R12\ \frac{\ll G_j,\sigma_j \gg \xrightarrow{\alpha^*} \ll G'_j,\sigma'_j \gg}{\|^j_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\alpha^*} \|_{i\in I} \ll G'_i,\sigma'_i \gg}\ \begin{array}{l} \ll G'_i,\sigma'_i \gg = \ll G_i,\sigma_i \gg \\ \forall i \neq j, \alpha^* \in Trans \cup \{\mu\} \end{array}$$

$$R13\ \frac{\ll G_j,\sigma_j \gg \xrightarrow{c!} \ll G'_j,\sigma'_j \gg \qquad \ll G_k,\sigma_k \gg \xrightarrow{c?(\sigma)} \ll G'_k,\sigma'_k \gg}{\|^j_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\mu} \|_{i\in I} \ll G'_i,\sigma'_i \gg}\ \begin{array}{c} \ll G'_i,\sigma'_i \gg = \ll G_i,\sigma_i \gg \\ \forall i \notin \{j,k\} \subseteq I \end{array}$$

$$R14\ \frac{\ll G_j,\sigma_j \gg \xrightarrow{c?(\sigma)} \ll G'_j,\sigma'_j \gg \qquad \ll G_k,\sigma_k \gg \xrightarrow{c!} \ll G'_k,\sigma'_k \gg}{\|^j_{i\in I} \ll G_i,\sigma_i \gg \xrightarrow{\mu} \|_{i\in I} \ll G'_i,\sigma'_i \gg}\ \begin{array}{c} \ll G'_i,\sigma'_i \gg = \ll G_i,\sigma_i \gg \\ \forall i \notin \{j,k\} \subseteq I \end{array}$$

$$R15\ \frac{\ll N,\sigma \gg \xrightarrow{\alpha} \ll G,\sigma' \gg}{\ll rec\ X.N,\sigma \gg \xrightarrow{\alpha} \ll G[rec\ X.N/X],\sigma' \gg} \qquad R16\ \frac{\ll P,\sigma \gg \longrightarrow_p \ll G,\sigma' \gg}{\ll rec\ X.P,\sigma \gg \longrightarrow_p \ll G[rec\ X.P/X],\sigma' \gg}$$

表 3：进程代数的操作语义。

• 规则 R3 表明，变换应用于当前状态，而新状态恰好就是变换的结果。

• 规则 R4 指出，如果对状态应用的条件成立，并且后面跟着一个非确定性过程，该过程能够通过执行变换或通信动作而演化，那么整个过程就通过执行该动作而演化。规则 R4' 表明，如果条件成立且后面跟着一个概率过程，那么整个过程会通过执行内部动作而演化，但不会立即执行概率动作（这样可以避免从同一配置同时启用非确定性步和概率步）。在这两条规则中，如果条件不成立，过程就不会演化。

• 规则 R5 是非确定性选择算子的常规规则：非确定性地选择一个动作来执行。具有多个选择的过程只能按照表示这些选择的任一过程自身能够演化的方式演化。因此，如果其中一个过程以一个不成立的条件开头，那么进行选择的过程不会因选择成为该过程而陷入停滞，因为该过程无法演化。[4]

• 规则 R6 是概率选择算子的常规规则。进程通过概率选择来演化。

• 规则 R7-R10 表示并行组合的第一步。在这一步中，并行组合中的某个进程"赢得竞争"，从而成为第二步中执行的进程。为此，只有关联进程能够继续演化的配置才能赢得竞争。因此，在 R7 中，j 配置必须通过变换或内部动作演化。在 R8 中，获胜配置能够通过概率转移演化。最后，在 R9 和 R10 中，j 配置必须

[4]注意，如果规则 R4 仅允许在 $cond$ 成立时"移除"进程中的 $[cond]$ 前缀 $[cond]N$ when（而不是像实际情况那样将 N $evolve$ 变为 N'），那么通过应用规则 R5，进程 $R ::= [cond]a?(x) + [cond]b?(x)$ 可以在一步之内演化为 $Q ::= a(x)?$ 或 $Q' ::= b(x)?$，前提是 $cond$ 成立。特别地，如果该进程 R 与进程 $a!$ 并行组合，那么当进程 R 演化 $in\ particular$ 为 Q'时，该系统就会陷入阻塞：进程 $a!$ 只能通过通道 a发送消息，而 Q' 只能通过通道 b接收消息。实际的规则 R4 避免了这一问题，因为进程 R 不能 $evolve$演化为 Q或 Q'。由于 R5 允许进行其任一选择所允许的任何演化，通过将 R4 应用于这些选择，进程 R 可以像 Q 所做的那样 $perform$，或者执行 Q' 所做的操作（这与 $becoming\ Q$ 或 Q' 本身有很大不同）。这样，R $simultaneously$ 就同时启用了两个选择 $a?(x)$ 和 $b?(x)$。特别要注意，进程 R 等价于 $[cond](a?(x) + b?(x))$。

能够参与通信，既可以作为发送方，也可以作为接收方。由于通信是同步的，发送方和接收方都必须准备好进行通信。因此，配置 j 只有在存在某个配置 k，使其能够与之进行通信，并且该配置 k 承担相反的通信角色时，才能赢得竞争。

• 规则 R11-14 表示并行组合的第二步。在所有这些规则中，我们都从一个并行配置开始，其中已经选定了将首先执行的配置。如果是概率配置赢得了竞争，则执行概率决策，并返回到一个非确定性并行配置，等待选定下一个要执行的配置（R11）。如果是非确定性配置赢得了竞争，则必须区分它是否为发送或接收状态的动作。如果既不是这两种动作中的任何一种，则该配置执行该动作，并像之前一样，返回到一个并行配置，等待选定下一个要运行的配置（R12）。另一方面，如果它是发送或接收状态的动作，那么要执行该动作，还必须执行其逆动作。在这种情况下，“赢得竞争”的配置以及与执行逆通信动作的进程相关联的配置都将发生演化（R13 和 R14）。

• 规则 R15 定义了递归非确定性进程的演化。如果进程 N 通过执行某个动作 α G演化为 G，那么递归进程 $(recX.N)$ 将通过执行该动作演化为 $G[recX.N/X]$ 。也就是说，它执行动作 α后演化为进程G ，其中变量 X 的所有出现位置都替换为进程 $(recX.N)$。

• 规则 R16 定义了递归概率进程的演化。如果进程 P通过执行概率动作演化为 G ，那么递归进程（$recX.P$）将以与之前相同的含义演化为 $G[recX.P/X]$ 。

4 蚁群优化细粒度并行化的形式化

下面我们将形式化并行实现的蚁群优化算法。细粒度实现和粗粒度实现都将使用 PA^{2}CO 进行描述。本节将介绍细粒度方法。

我们将首先对蚁群系统的两个版本进行规范：第一个版本包含若干个进程，其中每个进程代表一只蚂蚁，另有一个代表图的进程；所有进程并行执行，因此每只蚂蚁在每次迭代中完成一条路径。第二个版本则在前一个版本的基础上进行了修改，使同一只蚂蚁能够在一次迭代中执行多条路径，以期在部分蚂蚁速度较快时提高实现效率。在完全同质环境中，例如典型的基于GPU的实现，这种第二种方法用处不大。然而，对于运行蚁群优化的分布式或云计算实现而言，如果环境的异构性更强，且预期处理器的速度和内存可用性差异很大，那么这种方法就很合适。事实上，当所需的并行化级别超过任何一个我们能够承担使用的并行计算机的处理器数量时，这可能是唯一可行的方法。

之后，我们将讨论 $\mathcal{MM}$蚁群系统的规范，正如我们将看到的，它与蚁群系统的规范差异甚微。尽管这两个版本之间的差异很小，分别介绍它们仍然很有意义，因为蚁群系统是最早受到关注的蚁群优化算法，能够清晰地说明这类算法的工作原理，而 $\mathcal{MM}$蚁群系统则是蚁群优化中应用最广泛的版本。本节最后，我们将像介绍蚁群系统那样，在同样两种情况下介绍蚁群系统的规范。

在本节剩余部分，我们将认为 TSP 的图具有 n个 $I = \{1, \cdots, n\}$顶点，$I = \{1, \cdots, n\}$，数据 $= \mathbb{R}^+ \cup$列表$(\mathbb{R}^+) \cup$列表$($列表$(\mathbb{R}^+))$（其中，列表(A) 表示由 A 的元素构成的列表集合），以及 $0 \in \mathbb{R}^+$。

4.1 使用 m 只蚂蚁的细粒度蚁群系统形式化

我们首先对蚁群系统算法进行建模。设 $m \geq 1$ 为蚂蚁的数量。对于这种蚁群系统的细粒度并行化，我们将考虑两类进程：

- 一个描述蚂蚁行为的进程（将有 m 个此类进程），以及

- 另一个则描述图的行为。

表4展示了所需的变量、变换和条件集合。它们是构建其余各个案例的基础；在每个蚁群优化算法的形式化过程中，我们将重用其中的大部分。后续版本中新增的内容将在相应的变量、变换和条件表中以粗体标出。

变量	$= V_1 \cup V_2 \cup V_3$ 其中:
	$V_1 = \{$迭代次数、等待、***final***、已通知$\}$
	$V_2 = \{$路径$_k$、边$_k$，$L_k \mid 1 \leq k \leq m\}$
	$V_3 = \{\tau_{i,j} \mid i, j \in I, i \neq j\}$
Trans	$= T_1 \cup T_2 \cup T_3$ 其中:
	$T_1 = \{$***newIt***、***trailUpd***、***reset***、***unblock***、***finish***$\}$
	$T_2 = \{$阻塞$_k$，通知$_k \mid 1 \leq k \leq m\}$
	$T_3 = \{$移动$^k_{i,\ j} \mid 1 \leq k \leq m, i, j \in I, i \neq j\}$
$Cond$	$= C_1 \cup C_2$ 其中:
	$C_1 = \{$***end***，***end***$_A$，完成它，全部通知$\}$
	$C_2 = \{$完成$_k$，等待$_k$，已通知$_k \mid 1 \leq k \leq m\}$

表4：含有 m 只蚂蚁的蚁群系统的变量、变换和条件集合。

变量 $numIt \in \mathbb{R}^+$ 将记录算法目前已完成的迭代次数。我们认为，当所有蚂蚁都完成一条哈密顿路径时，一次迭代就结束了。在我们的建模中，图会阻止已经完成一圈的蚂蚁，让它们等待其他蚂蚁。因此，只有在所有蚂蚁都完成后才会更新轨迹。为了指定这些阻塞，我们引入变量 $waiting \in List(\mathbb{R}^+)$，用于保存每个时刻被阻塞的蚂蚁列表；还引入变换 $block_k$，将蚂蚁 a_k 加入等待列表。变量 $final \in \{0, 1\}$ 用于指示算法是否已结束：若结束则为 1，否则为 0。图会将此信息发送给蚂蚁，使它们停止搜索。变量 $notified \in List(\mathbb{R}^+)$将保存已收到此信息的蚂蚁列表。此外，对于每个$k \in \{1, \cdots, m\}$，我们引入变量 $path_k \in List(\mathbb{R}^+), edges_k \in List(\mathbb{R}^+)$和 $L_k \in \mathbb{R}^+$。变量 $path_k$ 将保存第 k 只蚂蚁在当前迭代中已访问的顶点列表，$edges_k$ 将保存该蚂蚁已选边的列表，而 $L_k \in \mathbb{R}^+$ 将表示它目前这一圈的长度（或成本）。虽然可以从$edges_k$ 中获得这些信息，但

也可以从$path_k$中获得。为清晰起见，我们将它们分开。最后， $\forall i, j \in I, i \neq j$ 我们考虑变量 $\tau_{i,j} \in \mathbb{R}^+$ ，它将保存图中边 $e_{i,j}$ 上的信息素量。

$$
\begin{aligned}
newIt : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[numIt \mapsto numIt \ + \ 1] \\
trailUpd : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[\tau_{i,j} \mapsto r(\tau_{i,j})]
\end{aligned}
$$

$$
r(\tau_{i,j}) = (1-\rho)\cdot\tau_{i,j} + \Sigma_{k=1}^{m}\Delta\tau_{i,j}^{k} \text{ where } \Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if } e_{i,j} \in edges_k \\ 0 & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
reset : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[\overline{\sigma}]
\end{aligned}
$$

$$
\begin{aligned}
\text{where} \quad \overline{\sigma} : \ & Var && - \to Data \\
& path_k && \longrightarrow last(path_k) && \forall k \in \{1, \cdots, m\} \\
& edges_k && \longrightarrow [\,] && \forall k \in \{1, \cdots, m\} \\
& L_k && \longrightarrow 0 && \forall k \in \{1, \cdots, m\}
\end{aligned}
$$

解除阻塞： $St \longrightarrow St\sigma \longrightarrow \sigma$[等待 $\mapsto [\,]$]阻塞$_k$： $St \longrightarrow St$ $\sigma \longrightarrow \sigma$[等待 $\mapsto$等待： k]移动$_{i,\ j}^{k}$： $St \longrightarrow St\sigma \longrightarrow \sigma$[路径$_k \mapsto$路径$_k$： v_j][边$_k \mapsto$边： $e_{i,\ j}$][$L_k \mapsto L_k + d_{i,j}$]完成： $St \longrightarrow St\sigma \longrightarrow \sigma$[最终 $\mapsto 1$] 通知 $_k$： $St \longrightarrow St\sigma \longrightarrow \sigma$[已通知 $\mapsto$ 已通知： k]

表5：蚁群系统的变换，其中有 m 只蚂蚁。

变换的定义见表5。算法的每次迭代结束后，变换$newIt$会增加计数器 $numIt$，而变换 $trailUpd$ 会更新图中各边上的信息素量。因此，函数 r 根据第2.1节中的公式 (2)更新轨迹， ρ 是信息素的衰减系数，而 Q是程序员提供的常量。

即将开始下一次迭代时，我们将使用变换$unblock$解除所有蚂蚁的阻塞，并使用 $reset$ 重新初始化变量 $path_k$、$edges_k$和 L_k $\forall k \in \{1, \cdots, m\}$。请注意，在本文余下部分中，我们用$last(l)$ 表示列表 l 的最后一个元素，用$l{:}e$ 表示在末尾添加元素 e 后的列表 l 。

迭代开始后，我们将使用 移动$_{i,\ j}^{k}$ 使第 k 只蚂蚁a_k沿边 $e_{i,\ j}$ 移动，使用 $i \neq j$。当第 k 只蚂蚁完成一圈后，图将阻塞它，并通过变换阻塞$_k$将它放入等待 列表。最后，当算法结束时，图将通过变换 完成 ，把最终 的值更新为 1，并将此信息发送给每只蚂蚁，同时通过变换通知$_k$将其保存到已通知 列表中；该变换会把第 k 只蚂蚁添加到已通知的蚂蚁列表中。

$$complete_k(\sigma) = \begin{cases} True & \text{如果 } len(\sigma(path_k)) = n \\ False & \text{否则} \end{cases} \qquad wait_k(\sigma) = \begin{cases} True & \text{如果 } k \in \sigma(waiting) \\ False & \text{否则} \end{cases}$$

$$completeIt(\sigma) = \begin{cases} True & \text{如果 } len(\sigma(waiting)) = m \\ False & \text{否则} \end{cases} \qquad end_A(\sigma) = \begin{cases} True & \text{如果 } \sigma(final) = 1 \\ False & \text{否则} \end{cases}$$

$$notified_k(\sigma) = \begin{cases} True & \text{如果 } k \in \sigma(notified) \\ False & \text{否则} \end{cases} \qquad allNotif(\sigma) = \begin{cases} True & \text{如果 } len(\sigma(notified)) = m \\ False & \text{否则} \end{cases}$$

$$end(\sigma) = \begin{cases} True & \text{如果 } (\sigma(numIt) = maxIt) \text{ 或 } (len(\sigma(waiting)) = m \text{ 且 } \sigma(path_k) = \sigma(path_{k+1})\ \ \forall k \in \{1, \cdots, m-1\}) \\ False & \text{否则} \end{cases}$$

$$\forall \sigma \in St \text{ 且 } \forall k \in \{1, \cdots, m\}$$

表 6：含有 m 只蚂蚁的蚁群系统的条件。

条件的定义见表 6。$complete_k$条件将告知我们第 k 只蚂蚁是否已完成一圈，即$path_k$ 的长度是否与顶点数量 n匹配。$wait_k$ 条件将说明第 k 只蚂蚁是否被图阻塞，也就是说，对于每个 $k \in \{1, \cdots, m\}$，第 k 只蚂蚁是否位于$waiting$列表中。当所有蚂蚁都完成一圈时，即$waiting$ 列表的长度为 m时，$completeIt$条件为真。

如果满足以下两个选项之一，end 条件将为真：

1. $numIt$ 与最大迭代次数匹配，或 2. $waiting$的长度为 m，且$path_k = path_{k+1}$ $\forall k \in \{1, \cdots, m-1\}$。

此条件会告知图算法已完成，以便将变量 $final$ 的值改为 1 并将其发送给蚂蚁。然后，

蚂蚁将评估条件end_A；当变量 *final*为 1 时，该条件为真。当第 k 只蚂蚁收到 *final*的新值，即该蚂蚁位于列表 $notified$中时，条件 $notified_k$ 将为真。最后，当所有蚂蚁都已收到通知时，$allNotif$ 将为真。

蚂蚁 k 与图之间的通信通道记为 grA_k，也就是说，我们将考虑 m通信通道。请注意，PA²CO 允许在同一通道中同时执行多个发送和接收动作。不过，我们为每只蚂蚁使用一个通道，这样便于形式化表示：图在接收到某只蚂蚁的信息后需要执行哪一个 $block_k$ 变换。此外，还应注意，在该规范及其余版本中，进程将发送完整状态，因此我们将简单地使用 $c!$ 表示通过通道发送当前状态 c（省略参数）。

定义变量、变换和条件集合后，便可按照表 7 中的形式定义蚂蚁和图的进程。

第 k 只蚂蚁

$A^k = rec\ X.(grA_k?(x);([\textit{end}\ \text{条件}_A]\text{停止} + [\neg\textit{end}\ \text{条件}_A]A^k{}_{info}))$
$A^k_{info} = recY.(\oplus_{i,j\in I}[p^k_{i,j}]\text{移动}^k_{i,j};([\text{完成}_k]A^k_C + [\neg\text{完成}_k]Y))$
$A^k_C = grA_k!;X$

图

$G = rec\ X.(\Sigma^m_{k=1}[\neg\text{等待}_k]grA_k!;X + \Sigma^m_{k=1}grA_k?(x);\text{阻塞}_k;X + [\text{完成}It]G_C)$
$G_C = trailUpd;newIt;([end]\text{完成};G_F + [\neg end]\text{解除阻塞};reset;X)G_F = recY.$
$(\Sigma^m_{k=1}[\neg\text{已通知}_k]grA_k!;\text{通知}_k;Y + [\text{全部通知}]\text{停止})$

表 7：含有 m 只蚂蚁的蚁群系统进程代数。

因此，代表每只蚂蚁 $k \in \{1,\cdots,m\}$ 的进程是一个递归进程，它首先向图请求信息，以获取每条边上的信息素数量，$\tau_{i,\ j}$，并评估end_A条件。如果算法已完成，该进程便结束。否则，执行进程 A^k_{info}，并根据从图中获取的信息和蚁群系统公式，计算概率以决定移动到哪个顶点 $p^k_{i,\ j}$。选定下一条边 $e_{i,\ j}$后，该进程通过变换$move^k_{i,\ j}$更新列表路径$_k$ 和边$_k$ 以及变量 L_k。此时，该进程

评估完成$_k$条件。如果它已完成一圈，就将信息发送给图，并重新启动代表该蚂蚁的递归进程。反之，如果尚未完成一圈，则返回递归进程 A^k_{info} 以选择下一条边。

表示图的进程必须通过阻塞来管理所有蚂蚁的信息，并处理进程的结束。因此，它是一个能够执行以下三种不同操作的递归进程：

1. 将自身的信息发送给任何提出请求且未被阻塞的蚂蚁，也就是说，发送给本次迭代中尚未完成一圈的蚂蚁。然后重新启动递归进程。

2. 接收其中一只蚂蚁的信息，此时阻塞该蚂蚁，并重新启动递归进程。

3. 评估条件$completeIt$。如果结果为真，则表示所有蚂蚁都已完成一圈，因此更新轨迹，并执行变换$newIt$ 来更新计数器$numIt$。随后评估条件end，以判断算法是否结束。如果已结束，则进程执行变换$finish$，将变量$final$ 的值改为 1，并执行进程 G_F。否则，解除所有蚂蚁的阻塞，将它们重新启动，并使其返回递归进程的起始位置。

请注意，如果$completeIt$ 为真，那么选项1和选项2都不会为真。选项1不可能为真，因为所有蚂蚁都会被阻塞，因此对于所有 $k \in \{1, \cdots, m\}$，条件$\neg$等待$_k$都为假。选项2也不可能，因为如果所有蚂蚁都被阻塞，就没有蚂蚁能够发送信息。因此，当一次迭代完成后，只能执行选项3。

最后，进程 G_F 是一个递归进程，其目标是通知所有蚂蚁算法已经结束。为此，它会向蚂蚁发送图数据（并附上变量最终值的新值）；当所有蚂蚁都收到通知后，该进程便结束。

综上，假设 $M = \{1, \cdots, m\}$，我们可以将蚁群系统建模为：

$$\|_{k \in M} \ll A_k, \sigma_k \gg \| \ll G, \sigma_G \gg$$

其中 $\forall k \in M$ 我们有：

$\sigma_k:$	Var	$-\rightarrow$	$Data$	$\sigma_G:$	Var	$-\rightarrow$	$Data$
	$path_k$	$\longrightarrow$	$[v_{0,k}]$		$numIt$	$\longrightarrow$	0
	$edges_k$	$\longrightarrow$	$[\,]$		$final$	$\longrightarrow$	0
	L_k	$\longrightarrow$	0		$notified$	$\longrightarrow$	$[\,]$
					$waiting$	$\longrightarrow$	$[\,]$
					$\tau_{i,j}$	$\longrightarrow$	τ_0 $\forall i,j \in \{1,\cdots,n\}, i \neq j$

其中，$v_{0,\ k}$ 表示算法开始时第 k 只蚂蚁所在的顶点，τ_0 表示每条边上的初始信息素量。也就是说，σ_k和 σ_G 是变量具有初始值的状态。

4.2 有 m 只自由蚂蚁的细粒度蚁群系统的形式化

在这个版本的蚁群系统中，我们不再强制蚂蚁彼此等待。这样，如果一只蚂蚁被分配到较快的处理器上，我们就可以利用其速度，而不必让它等待分配到较慢处理器上的其他蚂蚁完成各自的一圈。为了保持蚁群系统的理念，我们将继续考虑 m 条路径来执行轨迹更新，尽管其中有几条路径对应同一只蚂蚁。

由于蚂蚁不再会被阻塞，我们将删除上一节中引入的所有与阻塞相关的变量、变换和条件，并保留表8中的变量集、变换集和条件集，同时分别加入表9和表10中定义的新变换和条件。请注意，我们用 列表*1* $++$ 列表*2* 表示按此顺序连接列表*1* 和列表*2* 后得到的列表。

既然并非所有蚂蚁都会参与每次迭代，图就需要控制自己发送的信息。假设它没有这样做。那么，如果它在算法的最后一次迭代中发送了 $m+1$ 次信息，那么当它收到该次迭代生成的第 m 条路径时，就会评估条件 end，以检查算法是否已结束，并开始通知蚂蚁停止搜索。然而，已经接收信息、但最后发送信息的蚂蚁会一直处于阻塞状态，直到图愿意接收这条信息；而我们并不希望表示图的进程在算法完成后仍接收状态，因为有用的信息可能会被覆盖。为此，我们引入变量 $infoSent \in \mathbb{N}$、变换 $infoAdded$ 以及条件 $maxInfo$。其思路是：

变量 $= V_1 \cup V_2 \cup V_3$ 其中:
$V_1 = \{$迭代次数, ***final***, 路径, 边, 代价, 已通知, ***infoSent***$\}$
$V_2 = \{$路径$_k$, 边$_k$, $L_k \mid 1 \leq k \leq m\}$
$V_3 = \{\tau_{i,j} \mid i, j \in I, i \neq j\}$
Trans $= T_1 \cup T_2 \cup T_3$ 其中:
$T_1 = \{$***newIt***、***trailUpd***、完成、***resetLoop***、***infoAdded***$\}$
$T_2 = \{$***reset***$_k$,通知$_k$, ***addPath***$_k \mid 1 \leq k \leq m\}$
$T_3 = \{$移动$^k_{i,j} \mid 1 \leq k \leq m, i, j \in I, i \neq j\}$
$Cond = C_1 \cup C_2$ 其中:
$C_1 = \{$***end*** 条件，***end*** 条件$_A$，***completeIt***，全部通知，***maxInfo***$\}$
$C_2 = \{$完成$_k$，已通知$_k \mid 1 \leq k \leq m\}$

表8：变量集合 ，变换 ，以及蚁群系统中自由蚂蚁。

$infoSent$ 记录图发送信息的次数。因此，当它向一只蚂蚁发送信息时，会执行变换$infoAdded$来递增该计数器；并且只有在$infoSent$小于 m 时才能发送信息，这一点将由条件 $maxInfo$进行检查。

另一方面，由于现在同一只蚂蚁可以在同一次迭代中生成多条路径，图需要保存与这些路径相关的信息，以免蚂蚁重新启动时这些路径被清除。为此，我们引入变量 路径 $\in$ 列表($\mathbb{R}^+$)、边 $\in$ 列表($\mathbb{R}^+$) 和 代价 $\in$ 列表($\mathbb{R}^+$)，以及变换 $resetLoop$和$addPath_k$，每个 $k \in \{1, \cdots, m\}$ 对应一个。这样，每当一只蚂蚁 k 完成一圈时，路径$_k$、边$_k$ 和 L_k 将分别通过变换 $addPath_k$ 添加到列表 路径、边 和 代价 中。另一方面，$resetLoop$是一个变换，用于在每次新迭代开始时重置这些变量。

此外，由于蚂蚁不会彼此等待，它们不会同时全部重新启动；相反，图会在它们完成一圈时重新启动它们。因此，我们将变换$reset$ 替换为变换$reset_k$ ，每个$k \in \{1, \cdots, m\}$对应一个。

最后，由于信息的组织方式发生了变化，因此有必要重新定义变换 $trailUpd$ 中用于更新一条边的信息素数量的函数 r 。也就是说，现在我们不再使用 m变量路径$_k$、边$_k$ 和 L_k，而是使用列表路径、边 和代价。在表9中，

$\textit{infoAdded}\colon\ St \longrightarrow St\sigma \longrightarrow \sigma[\textit{infoSent} \mapsto \textit{infoSent} + 1]\textit{resetLoop}\colon St \longrightarrow St\sigma \longrightarrow \sigma[\text{路径} \mapsto [\,]][\text{边} \mapsto [\,]][\text{代价} \mapsto [\,]][\textit{infoSent} \mapsto 0]$

$\textit{addPath}_k\colon\ St \longrightarrow St \quad \forall k \in \{1, \cdots, m\}\sigma \longrightarrow \sigma[\text{路径} \mapsto \text{路径}'][\text{边} \mapsto \text{边}'][\text{代价} \mapsto \text{代价}'\text{路径'} = \text{路径} ++\text{路径}_k,\ \text{边}' = \text{边} ++\text{边}_k,\ \text{代价}' = \text{代价} ++L_k\textit{reset}_k\colon\ St \longrightarrow St\sigma \longrightarrow \sigma[\text{路径}_k \mapsto \textit{last(path}_k\textit{)}][\text{边}_k \mapsto [\,]][L_Q \mapsto 0]r(\tau_{i,j}) = (1-\rho)\cdot\tau_{i,\ j} + \Sigma_{k=1}^{m}\Delta\tau_i^k,\ {}_j\text{其中}\ \Delta\tau_i^k,$

$$_j = \begin{cases} \frac{Q}{costs[k]} & \text{如果}\ e_{i\,j} \in [k]\text{，否则为边} \\ 0 \end{cases}$$

表9：包含 m 只自由蚂蚁的蚁群系统变换。

$$maxInfo(\sigma) = \begin{cases} True & \text{if } \sigma(infoSent) < m \\ False & \text{otherwise} \end{cases}$$

$$\forall \sigma \in St \text{ and } \forall k \in \{1, \cdots, m\}$$

表10：包含 m 只自由蚂蚁的蚁群系统条件。

记法 边$[k]$ 和 代价$[k]$ 分别表示列表 边和 代价的位置， Q 是程序员给定的常量。注意，现在轨迹不会根据 m 蚂蚁更新，而是根据已生成的 m 路径更新；这些路径可能属于同一只蚂蚁，也可能属于多只蚂蚁，但不一定属于所有蚂蚁。此外，如果列表完成*It* 的长度为 m，那么条件路径 现在将为真。

蚂蚁和图的进程定义见表11。代表蚂蚁的进程行为保持不变。发生变化的是代表图的进程行为。这是一个递归进程，可以执行以下三种选项:

第 k 只蚂蚁

$A^k = rec\ X.(\mathit{grA}_k?\ \ (x)\ ;\ \ ([\mathit{end}\ 条件_A]停止\ +\ [\neg \mathit{end}\ 条件_A]A^k{}_{info}))$

$A^k_{info} = recY.(\oplus_{i,j \in I}[p^k_{i,j}]移动^k_{i,j};([完成_k]A^k_C \quad + \ [\neg 完成_k]Y))$

$A^k_C = \mathit{grA}_k!;X$

图

$G = rec\ X.([\neg \mathit{maxInfo}]G_E\ +\ G_R\ +\ [完成\mathit{It}]G_C)$

$G_E = \Sigma^m_{k=1}\mathit{grA}_k!\ ;\mathit{infoAdded};\ X$

$G_R = \Sigma^m_{k=1}\mathit{grA}_k?(x);添加路径_k;\mathit{reset}_k;\ X$

$G_C = \mathit{trailUpd};</style>newIt;([\mathit{end}\ 条件]完成;\ G_F \quad + \ [\neg \mathit{end}\ 条件]\mathit{resetLoop};\ \ X)$

$G_F = recY.(\Sigma^m_{k=1}[\neg 已通知_k]\mathit{grA}_k!;</style>通知;Y \quad + \ [全部通知]停止)$

表11：含 m 只自由蚂蚁的蚁群系统进程代数。

1. 运行进程 $[\neg maxInfo]G_E$。也就是说，如果信息尚未发送满 m 次，且某只蚂蚁向图请求信息，那么图就会发送当前状态，并返回递归进程的起始位置。

2. 运行进程 G_R。当某只蚂蚁 k 向图发送信息时，就会发生这种情况。如果发生这种情况，那一定是因为该蚂蚁完成了一圈，因此与这条路径相关的信息会通过变换$addPath_k$添加到列表 路径、边和代价中。随后，变量路径$_k$、边$_k$和 L_k 会重新初始化，使蚂蚁开始一条新路径。请注意，蚂蚁一旦重新启动自身的递归进程并向图请求信息，就会立即获得这些信息。最后，递归进程会重新启动。

3. 运行进程 $[completeIt]G_C$。该进程会评估条件$completeIt$，以确定一次迭代是否已完成。如果已完成，则使用变换$trailUpd$更新轨迹，并将计数器$numIt$增加$newIt$。随后评估条件 end，以确定算法是否已结束。如果已结束，则执行变换$finish$和进程 G_F；该进程与先前版本的蚁群系统中的进程相同，用于告知蚂蚁不要继续搜索。否则，将重新启动递归进程。

请注意，如果一次迭代已完成，则只有选项3能够执行。确实，如果一次迭代已完成，就意味着列表$paths$ 的长度为 m，因此图必须已发送其信息 m 次，并且条件 $[\neg maxInfo]$ 为假。此外，由于我们不允许信息发送超过 m 次，如果列表$paths$ 的长度为 m ，那一定是因为所有收到信息的蚂蚁都已将信息发送回来，因此没有蚂蚁能够再向图发送信息，选项2也不会执行。

综合以上内容，假设 $M = \{1, \cdots, m\}$，我们可以将包含 m只自由蚂蚁的蚁群系统建模如下：

$$\|_{k \in M} \ll A_k, \sigma_k \gg \| \ll G, \sigma_G \gg$$

其中 $\forall k \in M$

$\sigma_k:$	Var	$\longrightarrow$	$Data$	$\sigma_G:$	Var	$\longrightarrow$	$Data$
	$path_k$	$\longrightarrow$	$[v_{0,k}]$		$numIt$	$\longrightarrow$	0
	$edges_k$	$\longrightarrow$	$[\,]$		$final$	$\longrightarrow$	0
	L_k	$\longrightarrow$	0		$paths$	$\longrightarrow$	$[\,]$
					$edges$	$\longrightarrow$	$[\,]$
					$costs$	$\longrightarrow$	$[\,]$
					$infoSent$	$\longrightarrow$	0
					$notified$	$\longrightarrow$	$[\,]$
					$\tau_{i,j}$	$\longrightarrow$	$\tau_0 \ \forall i, j \in \{1, \cdots, n\}, i \neq j$

4.3 细粒度蚁群系统的形式化

要对标准蚁群系统和自由蚂蚁方法进行建模，只需修改轨迹更新中的函数 r ，使其考虑最优路径蚂蚁的精英策略，以及第2.2节中讨论的信息素量的上下限，即可使用以上所有内容。

4.4 包含 m 只蚂蚁的细粒度蚁群系统的形式化

现在我们介绍蚁群系统的这一版本。变量、变换和条件见表12。

对于蚁群系统（ACS），需要修改算法的行为，使其执行局部更新。因此，代表每只蚂蚁的进程将在图中移动，并检查算法是否已完成。图将负责执行局部更新，并判断蚂蚁是否已走完一圈；若是，则将执行离线 更新，使用

变量 $= V_1 \cup V_2 \cup V_3$ 其中:
$V_1 =$ {迭代次数、等待、最终、已通知}
$V_2 =$ {路径$_k$、边$_k \mid 1 \leq k \leq m$}
$V_3 = \{\tau_{i,j} \mid i, j \in I, i \neq j\}$
Trans $= T_1 \cup T_2 \cup T_3$ 其中:
$T_1 =$ {***newIt***、***trailUpd***、***reset***、解除阻塞}
$T_2 =$ {局部更新$_k$、阻塞$_k$、通知$_k \mid 1 \leq k \leq m$}
$T_3 =$ {移动$^k_{i,j} \mid 1 \leq k \leq m, i, j \in I, i \neq j$}
$Cond = C_1 \cup C_2$ 其中:
$C_1 =$ {***end*** 条件，***end***$_A$，完成它，全部通知}
$C_2 =$ {完成$_k$，等待$_k$，已通知$_k \mid 1 \leq k \leq m$}

表12：蚁群系统中含 m 只蚂蚁时的变量、变换和条件集合。

根据公式 (4) 修改后的变换$trailUpd$。此外，概率将按照 $\mathcal{MM}$蚁群系统（AS）中的方式计算。

$$
\begin{aligned}
localUpd_k : St &\longrightarrow St \quad \forall k \in \{1, \cdots, m\} \\
\sigma &\longmapsto \sigma[\tau_{i^*,j^*} \mapsto (1-\varphi) \cdot \tau_{i^*,j^*} + \varphi \cdot \tau_0] \\
&\text{where } e_{i^*,j^*} = last(edges_k)
\end{aligned}
$$

表13：有 m 只蚂蚁的蚁群系统的变换。

因此，我们添加变换集合 局部更新$_k$ ，并根据第 2.3 节的公式 (4) 对其进行定义，即要求 $e_{i,\ j}$ =最后(边$_k$)。也就是说，仅更新蚂蚁 k 的边$_k$ 中最后一条边上的信息素数量。

其余变量、变换和条件与第 4.1 节中有 m 只蚂蚁的蚁群系统版本相同。我们在表14中对该蚁群系统版本中的蚂蚁行为和图进行了建模。

在这种情况下，表示蚂蚁的进程相当简单。接收图的信息后，如果蚂蚁已经完成任务，就会停止执行；如果迭代尚未完成，则会专注于以概率方式遍历图。

第 k 只蚂蚁

$A^k = rec\ X.(grA_k?(x);(\ [end_A]$停止 $+\ [\neg end_A]A^k_{info}\ )$
$)A^k_{info} = \oplus_{i,j\in I}[p^k_{i,j}]$ 移动 $^k_{i,j}; grA_k!;\ X$

图

$G = rec\ X.(\ G_E\ +\ G_R\ +\ [$完成$It]G_C\ )G_E = \Sigma^n_{k=1}[\neg$等待$_k]grA_k!; X$
$G_R = \Sigma^m_{k=1} grA_k?(x);$ 局部更新$_k; ([$完成$_k]$阻塞$_k;\ X\ +\ [\neg$完成$_k];\ X)G_C =$
信息素轨迹更新; $newIt;$ 解除阻塞;$(\ [end]G_F\ +\ [\neg end]$重置$;\ X)$
$G_F = recY.(\ \Sigma^m_{k=1}[\neg$已通知$_k]grA_k!;$ 通知$_k; Y\ +\ [$全部通知$]$停止 $)$

表14：具有 m 只蚂蚁的蚁群系统的进程代数。

图进程是一个递归进程，可以执行三种选项：

1. 运行进程 G_E，该进程会向请求信息且未被阻塞的蚂蚁发送信息，

2. 运行进程 G_R，该进程接收来自一只蚂蚁的信息。在这种情况下，它执行相应的 局部更新$_k$ 变换，并评估<条件 完成$_k$。如果蚂蚁已完成一圈，则进程通过变换阻塞$_k$ 将其阻塞，然后重新开始。否则，它直接返回递归进程的开头，让蚂蚁继续构建其路径。

3. 运行进程 $[completeIt]G_C$，该进程首先评估条件$completeIt$，以确定是否必须执行信息素轨迹的离线 更新。因此，如果该条件为真，进程便执行以下变换：$trailUpd$ 以执行离线 更新；$newIt$ 以增加迭代计数器；以及 $unblock$，使蚂蚁再次能够接收信息。然后，评估条件 end，以查看算法是否已完成。如果没有，则对所有 $k \in \{1, \cdots, m\}$，变量路径$_k$、边$_k$和 L_k 将被重置，进程返回起点。请注意，图必须执行此重置，而且必须在此时执行。如果在评估*end* 条件之前执行，那么即使所有蚂蚁最终到达同一个顶点，该条件也可能为真，而它们未必都走过同一条路径，因此

只要它们最终都到达同一个顶点就足够了。此外，如果我们重置这些变量后算法结束，那么就无法保留蚂蚁的路径及其长度。另一方面，如果算法已经结束，则执行递归进程 G_F，通知所有蚂蚁，然后结束。

请注意，如果$completeIt$ 为真，那么与 AS 一样，该进程将无法执行 option1 或 option2。

因此，考虑到 $M = \{1, \cdots, m\}$，蚁群系统算法可形式化如下：

$$\|_{k \in M} \ll A_k, \sigma_k \gg \| \ll G, \sigma_G \gg$$

其中， σ_G 和 σ_k 与第 4.1 节中定义的相同。

4.5 使用 m 个自由蚂蚁的细粒度蚁群系统形式化

与从原始蚁群系统转变为带有自由蚂蚁的蚁群系统的情况一样，在此版本的蚁群系统中，我们也需要控制发送的信息。不过，现在我们不会在每次迭代中发送信息 m 次，而是发送 $m \cdot n$ 次，也就是说，每当一只蚂蚁需要选择一条新边时，就发送一次。

变量 $= V_1 \cup V_2 \cup V_3$ 其中：$V_1 = \{$迭代次数、***final***、路径、边、代价、已通知、搜索、***infoSent***$\}V_2 = \{$路径$_k$、边$_k$、 $L_k \mid 1 \le k \le m\}$ $V_3 = \{\tau_{i,j} \mid i, j \in I, i \neq j\}$转移 $= T_1 \cup T_2 \cup T_3$ 其中：$T_1 = \{$***trailUpd***、***newIt***、***resetLoop***、***infoAdded***$\}T_2 = \{$局部更新$_k$、***reset***$_k$、通知$_k$、添加搜索$_k$、擦除搜索$_k \mid 1 \le k \le m\}T_3 = \{$***move***$^k_{i,\ j} \mid 1 \le k \le m$、 i、 $j \in I$、$i \neq j\}$条件 $= C_1 \cup C_2$ 其中：$C_1 = \{$***end***、***end***$_A$、***completeIt***、***allNotif***、***maxInfo***$\}C_2 = \{$完成$_k$、已通知$_k$、搜索中$_k \mid 1 \le k \le m\}$

表15：含 m 只自由蚂蚁的蚁群系统的变量集、变换和条件集。

表15给出了此情形下的变量、变换和条件集合

。对此需要考虑的变化不能简单地设定为信息只应发送 $m \cdot n$ 次，因为这无法保证信息总是发送给相同的 m 只蚂蚁。这里的“蚂蚁”指某只蚂蚁的迭代。请记住，当蚂蚁彼此无需等待时， m 条路径不一定由 m 只不同的蚂蚁构建，也可能来自同一只蚂蚁。为解决这个问题，我们引入变量 搜索、变换集合 添加搜索$_k$ 和擦除搜索$_k$，以及针对每个 $k \in \{1, \cdots, m\}$ 的条件 搜索中$_k$。因此，搜索 将记录已经开始走一圈的蚂蚁列表，而 搜索中$_k$ 将告知我们第 k 只蚂蚁是否在搜索列表中。这些新变换和条件分别定义于表16和表17。

$$
\begin{aligned}
addSearch_k : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[search \mapsto search +\!+ k] \\
eraseSearch_k : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[search \mapsto remove(k, search)] \\
resetLoop : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma' \\
\sigma' = \sigma[paths \mapsto [\,]][edges \mapsto [\,]][costs \mapsto [\,]][infoSent \mapsto 0][search &\mapsto [\,]]
\end{aligned}
$$

表16：具有 m 个自由蚂蚁的蚁群系统的变换。

$$
searching_k(\sigma) = \begin{cases} True & \text{if } k \in search \\ False & \text{otherwise} \end{cases}
$$

$$
\forall \sigma \in St \text{ and } \forall k \in \{1, \cdots, m\}
$$

表17：具有 m 自由蚂蚁的蚁群系统的条件。

这样，当一只蚂蚁开始搜索时，我们会通过变换添加搜索$_k$ 将它加入搜索列表，并通过变换$infoAdded$将计数器$infoSent$加一。当一只蚂蚁完成一圈后，我们会通过变换擦除搜索$_k$将它从列表中移除，其中函数$remove(elem,l)$ 会从列表 l中移除项目$elem$ 。因此，搜索会告诉我们

始终有哪些蚂蚁正在遍历图，而变量$infoSent$会告诉我们，在本次算法迭代中，有多少只蚂蚁（或某只蚂蚁进行了多少次迭代）执行了搜索。此外，在这种情况下，变换$resetLoop$也必须重新启动搜索列表。

第 k 只蚂蚁

$$A^k = rec\ X.(\ \ grA_k?(x);(\ \ [end_A]stop\ +\ [\neg end_A]A^k_{info}\ ))$$
$$A^k_{info} = \oplus_{i,j\in I}[p^k_{i,j}]move^k_{i,j};\ grA_k!;\ X$$

图

$$G = rec\ X.(G_E\ +\ G_R\ +\ [completeIt]G_C)$$
$$G_E = \Sigma^m_{k=1}([searching_k]grA_k!;\ X + [\neg searching_k]G_{EN})$$
$$G_{EN} = [maxInfo]X + [\neg maxInfo]grA_k!;\ addSearch_k;\ infoAdded;\ X$$
$$G_R = \Sigma^m_{k=1}grA_k?(x);\ localUpd_k([complete_k]reset_k;\ eraseSearch_k;\ X + [\neg complete_k]X)$$
$$G_C = trailUpd;\ newIt;\ ([end]G_F\ +\ [\neg end]resetLoop;\ X)$$
$$G_F = recY.(\Sigma^m_{k=1}[\neg notified_k]grA_k!;\ notify_k;\ Y + [allNotif]stop)$$

表18：含有 m 只自由蚂蚁的蚁群系统进程代数。

表18定义了蚂蚁和图的进程。蚂蚁的进程与先前版本相同，因为它们专门负责选择边，并立即将信息发送给图，直至算法结束。

另一方面，表示图的进程是一个递归进程，可以执行三种操作：

1. 运行进程 G_E，该进程负责向蚂蚁发送信息。如果第 k 只蚂蚁想从图中获取信息，那么进程 G_E 会评估条件 搜索中 ，以确认这只蚂蚁是否已经在图中移动。如果是，图就向它发送其状态，并重新启动初始递归进程。另一方面，如果蚂蚁打算开始一圈，则执行进程 G_{EN} 。该进程会评估条件 $maxInfo$，以确认将在本次迭代中构建路径的 m 只蚂蚁（或某只蚂蚁的迭代）是否已经选定；如果是，则重新启动递归进程。另一方面，如果仍可向新蚂蚁发送信息，则将信息发送给该蚂蚁，使用变换添加搜索将其加入列表搜索 ，增加计数器 $infoSent$ 的值，并重新启动递归进程。

2. 运行进程 G_R，它从蚂蚁 k接收信息。在此

情况下，它执行相应的局部更新$_k$ 变换，并判断条件完成$_k$。如果蚂蚁已经完成一圈，进程便通过相应的$reset_k$ 变换重置其变量，通过擦除搜索$_k$ 变换将蚂蚁从搜索列表中移除，然后重新开始。否则，它只需返回递归进程的开头，让蚂蚁继续构建其路径。

3. 运行进程 $[completeIt]G_C$，该进程首先评估条件 $completeIt$ ，以确定是否需要对信息素轨迹执行离线更新。因此，如果满足此条件，进程便执行变换 $trailUpd$来执行离线更新，并执行$newIt$ 以增加迭代计数器。之后，会检查条件 end，以确定算法是否已结束。否则，执行变换 $resetLoop$ ，以重置迭代变量并重新启动递归进程。另一方面，如果算法已结束，则执行递归进程 G_F ，通知所有蚂蚁，然后结束。

请注意，如果一轮迭代已完成，进程将无法执行选项 1 或选项 2。事实上，如果一轮已完成，就意味着所有正在搜索的蚂蚁都已完成这一轮，因此在评估完成$_k$ 之后，进程已通过 添加搜索 变换将它们从搜索列表中移除。选项 2 也不可行，因为如果所有蚂蚁都已完成，它们就没有信息可发送。

综上，考虑到 $M = \{1, \cdots, m\}$，我们可以将包含m只自由蚂蚁的蚁群系统建模为:

$$\|_{k \in M} \ll H_k, \sigma_k \gg \| \ll G, \sigma_G \gg$$

其中 $\forall k \in M$ 我们有， σ_k 的定义与前一种情况相同，而 σ_G 的定义也与之前相同，但加入了变量 搜索，其值为空列表。也就是说， σ_k 和 σ_G 是变量具有初始值的状态。

5 蚁群优化粗粒度并行化的形式化

粗粒度 并行化是指将规模较大的子群体分配给不同的处理器，并使它们之间的通信量较低。正如我们在第 2.4 节中所见，Manfrin 等人（2006）通过实证证明，能够取得最佳结果的蚁群优化实现方式，是使用变量的不同初始值，独立运行同一串行算法 $\mathcal{MM}$蚁群系统的多个副本。在这种实现中，会设定最大迭代次数，各算法副本彼此之间不进行任何通信地运行。待它们全部结束后，从不同副本给出的结果中选出最优解。由于这已被证明是效率最高的策略，我们将对蚁群系统、 $\mathcal{MM}$蚁群系统和蚁群系统（ACS）的这种并行化方法进行形式化。此外，为了说明如何对从最激进的细粒度方法到最激进的粗粒度方法这一范围内的任何实现进行形式化，我们还将在本节末讨论如何对带有偶尔通信的粗粒度并行化进行形式化。

5.1 使用 p 次独立执行的粗粒度蚁群系统形式化

本节的思路是使用进程 C，为每个顺序蚁群系统算法副本指定形式，并以最大迭代次数作为停止条件。所有副本完成后，它们会将信息发送给进程 B；该进程将接收各副本的解，并选出其中最优解。请注意，由于现在指定的是串行算法，因此无需像前几节那样明确区分蚂蚁和图。

设 n 为旅行商问题中的顶点数， $p+1$ 为解决该问题可用的处理器数， m 为算法采用的蚂蚁数。设 $I = \{1, \cdots, n\}$、 $S = \{1, \cdots, p\}$和 $M = \{1, \cdots, m\}$。算法的每个副本 C_s 将与处理器 $s \in S$ 关联，而 B 将与处理器 $p+1$关联。

用于形式化所有 C_s $s \in S$ 和 B 副本的变量、变换和条件列于表19中。

变量 $solution_s \in$ 列表$(\mathbb{R}^+)$ 将保存副本 C_s 对所有 $s \in S$ 的解，而进程 B 将保存它从其他进

数据	$= \mathbb{R}^+ \cup List(\mathbb{R}^+)$
变量	$= V_1 \cup V_2 \cup V_3$ 其中:
	$V_1 = \{$迭代次数，解，结果$\}$
	$V_2 = \{$路径$_k$，边$_k, L_k \mid k \in M\}$
	$V_3 = \{\tau_{i,j} \mid i,j \in \{1,\cdots,n\}, i \neq j\}$
Trans	$= T_1 \cup T_2$ 其中:
	$T_1 = \{trailUpd\ reset,\ newIt,$ ***calculateBest***$\} \cup \{$***selBest***$_s,$ ***addSol***$_s \mid s \in S\}$
	$T_2 = \{$移动$^k_{i,j} \mid k \in M,\ i,\ j \in I,\ i \neq j\}$
$Cond$	$= \{$完成，结束条件，所有解$\}$

表19：用于并行化具有 p个独立执行的蚁群系统的粗粒度变量集、变换和条件。

程接收到的解，存储于 解 $\in$ 列表(列表($\mathbb{R}^+$)) 中，并将其中最佳的解存入 结果 $\in$列表($\mathbb{R}^+$)。

关于变换，$trailUpd$将采用第2.2节中讨论的、由具有最优路径的蚂蚁实施的精英策略，并限制信息素的数量。此外，我们添加变换$selBest_s\ \forall s \in S$，使每个算法副本在完成后都能从所有蚂蚁提供的路径中选出最优路径。表20定义了这一新变换，其中 $k^* \in M$ 是构建出更优路径的蚂蚁的索引，即 $k^* = argmin_{k \in M}\{L_k\}$。请注意，变量$solutions_s$ 被赋值为一个列表，其第一个元素是最优路径的成本，其余部分则恰好是最优路径。这将有助于我们比较所有独立执行的算法所得到的解。

$selBest_s : St$	$\longrightarrow St$	
σ	$\longrightarrow \sigma[solution_s \mapsto [L_{k^*}] ++ path_{k^*}]$	其中 $k^* = argmin_{k \in M}\{L_k\}$
$addSol_s : St$	$\longrightarrow St$	
σ	$\longrightarrow \sigma[solutions \mapsto solutions ++ solution_s]$	
$calculateBest : St$	$\longrightarrow St$	
σ	$\longrightarrow \sigma[result \mapsto solutions[k^*]]$	其中 $k^* = argmin_{k \in M}\{L_k\}$

表20： C_s 和 B的变换。

我们加入变换$addSol_s$，适用于所有 $s \in S$，并加入$calculateBest$，以便 B 选择最优解。这样，当它通过通道$sols_s$收到副本 s 的解时，就会使用变换 $addSol_s$将列表$solutions_s$ 添加到列表$solutions$中。收到所有解后，它

将使用变换$calculateBest$选出最优解，并将其赋给变量$result$。$addSol_s$、$\forall s \in S$和$calculateBest$的定义也列于表20，其中 $k^* \in M$ 是列表$solutions$ 中对应于最优解的索引。也就是说， $k^* = argmin_{k \in M}\{solutions[k][1]\}$，其中列表首个元素的索引视为1。当 B 收到所有解时，条件$allSolutions$为真，也就是说，列表$solutions$的长度为 m ，其形式化定义见表21。最后，条件$complete$ 用于告知每个算法副本一次迭代是否完成。请注意，由于这是一个串行算法，当蚂蚁的任一路径长度为 n 时（例如蚂蚁1），该条件为真。

$$allSolutions(\sigma) = \begin{cases} True & \text{if } len(\sigma(solutions)) = m \\ False & \text{otherwise} \end{cases}$$

$$complete(\sigma) = \begin{cases} True & \text{if } len(\sigma(path_1)) = n \\ False & \text{otherwise} \end{cases}$$

$$\forall \sigma \in St$$

表21：具有 p 次独立执行的蚁群系统粗粒度并行化条件。

利用这些集合，我们按照表 22 对进程 C_s、 $\forall s \in S$和 B 进行形式化。因此，C_s 是一个递归进程，它首先以概率方式将每只蚂蚁移动到一条边上。随后，它会评估完成条件，以判断一次迭代是否已完成。如果尚未完成，则重新启动递归进程以选择新的边。如果一次迭代已完成，则更新轨迹，递增迭代次数计数器，并评估 end 条件以判断算法是否已结束；如果已结束，则选出最优解，通过通道$sols_s$ 发送，并结束进程。另一方面，如果算法尚未结束，则重新启动递归进程，开始下一次迭代。

进程 B 接收其他进程的解，并将它们添加到列表solutions中，然后选出其中最优的解，将其存储在变量result中，最后结束。

综合以上所有内容，我们可以将蚁群系统的这一实现表示为并行模型：

$$\|_{s \in S} \ll C_s, \sigma_s \gg \| \ll B, \sigma_B \gg$$

第 s 个算法副本

$C_s = rec\ X.(\oplus_{i,j\in I}[p^1_{i,j}]move^1_{i,j}; \cdots; \oplus_{i,j\in I}[p^m_{i,j}]move^m_{i,j}; ([complete]C^s_C + [\neg complete]X))$
$C^s_C = trailUpd; newIt; ([end]selBest_s; sols_s!;\ stop + [\neg end]reset;\ X)$

进程 B

$B = rec\ X.(\ \Sigma_{s\in S} sols_s?(x); addSol_s; X\ +\ [allSolutions]calculateBest;\ stop)$

表 22：蚁群系统的粗粒度 形式化，包含 p 次独立执行。

其中，对于每个 $s \in S$，我们有：

σ_s :	Var	$- \to$	$Data$		σ_B :	Var	$- \to$	$Data$
	迭代次数	$\longrightarrow$	0			解	$\longrightarrow$	[ ]
	$\tau_{i,j}$	$\longrightarrow$	τ^s_0	$\forall i, j \in I, i \neq j$		结果	$\longrightarrow$	[ ]
	$path_k$	$\longrightarrow$	$[v^s_{0,k}]$	$\forall k \in M$				
	边 k	$\longrightarrow$	[ ]	$\forall k \in M$				
	L_k	$\longrightarrow$	0	$\forall k \in M$				

请注意，在算法的每个副本中，已初始化变量的值都不相同（或者说，不一定相同）。

5.2 使用 p 次独立执行的粗粒度 $\mathcal{MM}$蚁群系统形式化

这些 $\mathcal{MM}$蚁群系统实现中的 p 独立执行，其形式化方式几乎与上一节所述相同。我们只需修改轨迹更新中的函数 r ，以考虑具有最优路径的蚂蚁所采用的精英策略，以及第2.2节中讨论的信息素数量的上下限。

5.3 具有 p独立执行的粗粒度蚁群系统的形式化

在这一包含 p 个独立算法副本的场景中，蚁群系统的形式化与前述情况也非常相似，因为我们只需添加局部更新，并考虑用于计算概率的适当函数（参见第 2.3 节）。因此，此形式化所用的变量、变换和条件

与表19中的内容相同，另加此前在表13中定义的变换 局部更新$_k$，
$\forall k \in \{1, \cdots, m\}$。

第 s 个算法副本

$C_s = rec\ X.(\oplus_{i,\ j \in I}[p^1_{i,j}]move^1_{i,\ j};\ \cdots;\ \oplus_{i,\ j \in I}[p^m_{i,j}]move^m_{i,\ j};\ localUpd_1;\ .....;\ localUpd_m;\ ([complete]C^s_C\ \ + [\neg complete]X))$

$C^s_C = trailUpd;\ \ newIt;\ \ ([end]selBest_s;\ \ sols_s!;\ \ stop + [\neg end]reset;\ \ X)$

进程 B

$B = rec\ X.(\ \Sigma_{s \in S} sols_s?(x);\ \ addSol_s;\ \ X\ +\ [allSolutions]calculateBest;\ \ stop)$

表23：粗粒度蚁群系统的形式化，采用 p 独立执行。

在这种情况下，表示每个副本 C_s、 $s \in S$的进程，以及负责选择最终解的进程B，可如表23所示进行描述；其中，蚁群系统中引入的局部更新在每个进程 C_s 中执行，此时所有蚂蚁均已选择下一个顶点，而$trailUpd$ 则使用函数r ，该函数已在第 2.3 节中说明。蚁群系统的整体实现可建模为:

$$\|_{s \in S} \ll C_s, \sigma_s \gg \| \ll B, \sigma_B \gg$$

其中，对于每个 $s \in S$，我们有:

σ_s：	Var	$- \rightarrow$	$Data$		σ_B：	Var	$- \rightarrow$	$Data$
	迭代次数	$\longrightarrow$	0			解	$\longrightarrow$	$[\,]$
	$\tau_{i,j}$	$\longrightarrow$	τ^s_0	$\forall i, j \in I, i \neq j$		结果	$\longrightarrow$	$[\,]$
	$path_k$	$\longrightarrow$	$[v^s_{0,k}]$	$\forall k \in M$				
	边 $_k$	$\longrightarrow$	$[\,]$	$\forall k \in M$				
	L_k	$\longrightarrow$	0	$\forall k \in M$				

再次强调，在每个算法副本中，已初始化变量的值各不相同（或者不一定相同）。

5.4 粗粒度并行化与偶尔通信的形式化

在前文中，我们已经对最“激进”的<r>粗粒度</r>无通信方法进行了形式化。本节给出的形式化旨在说明如何对考虑

进程间偶尔通信的粗粒度并行化进行形式化。若设计者希望以同样的粗粒度构建规范，就应使用本文提供的工具补充所需的细节。

我们将考虑多个顺序AS副本（同样的思路也适用于 $\mathcal{MM}$AS 和蚁群系统），这一次，经过固定次数的迭代后，各副本将共享各自的轨迹（即信息素轨迹），并根据彼此的轨迹进行更新。我们将考虑一个通用函数 f^* 来执行此更新，以便设计者可以选择适用于预期方法的具体函数。此类共享函数的一些典型示例如下：

- 使用迄今为止获得最优解的副本的轨迹来更新轨迹，
- 使用各副本轨迹的平均值来更新轨迹，或者
- 使用加权平均值来更新轨迹，通常会优先考虑自身的轨迹或最优轨迹。

达到固定的最大迭代次数 $maxIt$ 后，每个副本都会结束。

我们将采用一种结合了细粒度 情形背后的思路与前述粗粒度 方法的逻辑，来形式化这一实现：与粗粒度 方法一样，其中包含 p 个顺序算法副本；此外还有一个进程 G^*，其行为类似于细粒度 方法中的图，但现在它充当一个“超图”，承载不同副本的图所包含的信息。该进程负责接收算法各副本的轨迹，并应用相应的 f^* 函数。它还负责结束其余进程。

设 n 为旅行商问题中的顶点数，$p+1$ 为解决该问题可用的处理器数，m 为算法考虑的蚂蚁数，u 为算法的每个副本在共享其轨迹之前执行的迭代次数。设 $I=\{1,\cdots,n\}$、$S=\{1,\cdots,p\}$ 和 $M=\{1,\cdots,m\}$。算法的每个副本 C_s 将与处理器 $s\in S$ 关联，而 G^* 将与处理器 $p+1$ 关联。

此形式化所用的变量、变换和条件列于表24。请注意，其中一些内容此前已在细粒度实现中定义（见第4节），但当时指的是蚂蚁（例如区块$_k$），而非副本。必要的调整和修改并不困难，因此此处省略具体细节。

变量 $= V_1 \cup V_2 \cup V_3$ 其中:
$V_1 =$ {迭代次数、等待、最终、已通知}
$V_2 =$ {路径$_k^s$、边$_k^s$、 $L_k^s \vert 1 \le k \le m$、 $1 \le s \le p$}
$V_3 = \{\tau_{i,j}^s \vert 1 \le s \le p, i, j \in I, i \ne j\}$
$\mathit{Trans} = T_1 \cup T_2 \cup T_3$ 其中:
$T_1 =$ {***newIt***、***trailUpd***$_s$、***trailUpd****、***reset***、解除阻塞、完成}
$T_2 =$ {阻塞$_s$、通知$_s \vert 1 \le s \le p$}
$T_3 =$ {移动$_{i,\ j}^k \vert 1 \le k \le m, i, j \in I, i \ne j$}
$Cond = C_1 \cup C_2$ 其中:
$C_1 =$ {***end***，***end***$_A$，完成，***completeIt***，全部通知，***shareIt***}
$C_2 =$ {等待$_s$，已通知$_s \vert 1 \le s \le p$}

表24：采用偶尔通信的蚁群系统粗粒度并行化所涉及的变量、变换和条件集合。

在此实现中，有 p 个图（算法的每个副本各对应一个图），因此每个图都有自己的轨迹变量，并且只更新自身的轨迹。这就是我们引入变换$trailUpd_s$ $\forall s \in S$的原因。此外，我们还加入变换$trailUpd^*$，用于将函数 f^*应用于轨迹。这两个变换在表25中作了形式化定义。

$$
\begin{aligned}
trailUpd_s : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[\tau_{i,j}^s \mapsto r(\tau_{i,j}^s)] \\
r(\tau_{i,j}^s) = (1-\rho)\cdot\tau_{i,j}^s + \Sigma_{k=1}^m \Delta\tau_{i,j}^{s,k} \text{ where } \Delta\tau_{i,j}^{s,k} &= \begin{cases} \frac{Q}{L_k} & \text{if } e_{i,j} \in edges_k \\ 0 & \text{otherwise} \end{cases} \\
trailUpd^* : St &\longrightarrow St \\
\sigma &\longrightarrow \sigma[\tau_{i,j}^s \mapsto f^*(s, \tau_{i,j}^1, \cdots, \tau_{i,j}^p)] \quad \forall s \in S
\end{aligned}
$$

表25：采用偶尔通信的蚁群系统粗粒度形式化的变换。

此外，我们添加条件 $shareIt$，用于告知每个副本是否需要向超图发送信息，也就是说，它是否已完成接下来的 u 次迭代，或者是否已达到最后一次迭代 $maxIt$ 且算法已经结束。此条件在表26中明确定义。

$$\textit{shareIt}(\sigma) = \begin{cases} \text{若 if } (\sigma(\text{迭代次数}) = \textit{maxInfo}) \text{ 或 } (\sigma(\text{迭代次数}) \bmod u \ = 0) \end{cases}$$

则为 ***True***，否则为 ***False***otherwise

$$\forall \sigma \in St \text{ and } \forall k \in \{1, \cdots, m\}$$

表26：带偶尔通信的蚁群系统粗粒度 形式化条件。

根据以上所有内容，副本和超图的进程定义见表 27。

第 s 个算法副本

$C_s = rec\ X.(channel_s?(x);\ ([end_A]stop + [\neg end_A]C^s{}_{info}))$
$C^s_{info} = recY.(\oplus_{i,\ j\in I}[p^1_{i,j}]move^1_{i,\ j};\cdots;\oplus_{i,\ j\in I}[p^m_{i,j}]move^m_{i,\ j};([complete]C^s_{send} + [\neg complete]Y)$
$C^s_{send} = trailUpd_s; newIt; reset; ([shareIt]channel_s!;\ X\ +\ [\neg shareIt]Y)$

超图

$G^* = rec\ X.(\Sigma^p_{s=1}[\neg wait_s]channel_s!; X + \Sigma^p_{s=1}\ channel_s?(x); block_s; X + [completeIt]G^*_C)$
$G^*_C = ([end]finish; G_F\ +\ [\neg end]trailUpd^*; unblock;\ X)$
$G^*_F = recY.(\Sigma^p_{s=1}[\neg notified_s]channel_s!;\ notify_s; Y\ +\ [allNotif]stop)$

表 27：粗粒度蚁群系统的偶尔通信形式化

与我们在标准蚁群系统的第 4.1 节中研究的情况类似，超图会将算法各个副本的轨迹发送给每个进程$C_s\ s \in S$，每个副本都会依次执行 u次算法迭代。完成这组迭代后，它们会将自己的轨迹发送给超图。每当超图收到某个副本的信息时，就会将其阻塞，以便其余副本也能完成各自的 u次迭代。当所有副本都完成这组迭代后，超图会通过变换$trailUpd^*$应用全局更新函数 f^*，解除所有副本的阻塞，并向它们发送新的轨迹。条件end 会通过将$numIt$与最大迭代次数进行比较，告知超图各个副本是否已完成算法。注意，它比较的是$numIt$与 $maxIt$；后者会被赋值为最后一个发送信息的算法副本的值。这不会造成问题，因为所有副本都会相互等待

在每组 u 次迭代中，因此它们发送给超图的$numIt$ 值都相同。当条件end 为真时，超图会通知各个副本它们都已完成，使它们停止执行，就像我们在标准蚁群系统的第 4.1 节中所做的那样。

最后请注意，我们省略了这样一个过程：所有副本将各自的解发送到超图，由超图像之前的粗粒度 并行化中那样选出最佳解。设计者需要根据预期采用的偶发通信方法，完成这部分的形式化，如前文所述。

蚁群系统的整体实现可建模为：

$$\|_{s\in S} \ll C_s, \sigma_s \gg \| \ll G^*, \sigma_{G^*} \gg$$

其中，对于每个 $s \in S$，我们有：

$$
\begin{array}{lllll}
\sigma_s: & Var & - \to & Data & \\
 & \text{迭代次数} & \longrightarrow & 0 & \\
 & \tau^s_{i,j} & \longrightarrow & \tau^s_0 & \forall i,j \in I, i \neq j \\
 & \text{路径}^s_k & \longrightarrow & [v^s_{0,k}] & \forall k \in M \\
 & \text{边}^s_k & \longrightarrow & [\,] & \forall k \in M \\
 & L^s_k & \longrightarrow & 0 & \forall k \in M
\end{array}
\qquad
\begin{array}{lll}
\sigma_{G^*}: & Var & - \to \quad Data \\
 & \text{等待} & \longrightarrow \quad [\,] \\
 & \text{已通知} & \longrightarrow \quad [\,] \\
 & final & \longrightarrow \quad 0
\end{array}
$$

同样，已初始化变量的值在算法的每个副本中通常各不相同。

6 讨论：规格比较

本节总结论文中介绍的所有蚁群优化变体，并重点说明各个变体的具体差异，以便相互比较。

在蚁群优化的细粒度规格中，所有蚂蚁都与图进程并行运行。本例中我们考虑了以下几种规范变体：

- 蚁群系统：在蚁群系统的细粒度规范中，当一只蚂蚁 k 完成其迭代时，图会通过变换 阻塞$_k$将其阻塞。此变换会将其加入 等待 变量，该变量用于存储已完成迭代的蚂蚁。图会评估一个条件，以确保不会向此列表中的任何蚂蚁发送信息，从而使这些蚂蚁无法开始下一轮

迭代。当所有蚂蚁都在等待 列表中时，图会解除对它们的阻塞，重置变量，然后算法继续进行下一轮迭代。

- $\mathcal{MM}$蚁群系统： $\mathcal{MM}$蚁群系统的细粒度规范与前一个类似，但引入了精英策略及相应的信息素轨迹更新机制。

- 蚁群系统：蚁群系统的细粒度规范与蚁群系统对应的规范类似，不同之处在于引入了蚂蚁移动时的局部更新机制。为了执行这些额外更新，图会在每只蚂蚁每次移动时接收其信息。因此，图进程负责统计迭代何时完成，以便执行 离线 更新。

- 带有自由蚂蚁的蚁群系统：在蚁群系统细粒度规范的这一替代版本中，速度较快的蚂蚁可以在每次算法迭代中遍历更多路径。为了使每次迭代都基于相同数量的新蚂蚁遍历，并确保蚂蚁不会执行不计入结果的任务，图进程会统计发送给蚂蚁的信息量，从而向所有蚂蚁发送预先设定数量的执行图遍历请求。

- $\mathcal{MM}$带有自由蚂蚁的蚁群系统：这是前一个变体的 $\mathcal{MM}$蚁群系统版本。

- 自由蚂蚁版 ACS：这是前两个变体的 ACS 版本。与对应的蚁群系统版本相比，该规范的特殊之处在于，现在我们有局部更新操作，这意味着每次迭代都要向每只蚂蚁发送 n 次数据：每当一只蚂蚁移动到另一个节点时，图就会向它发送新的信息素轨迹。请注意，仅发送 $n \cdot m$ 次数据的策略行不通，因为它无法保证数据会发送给已经开始移动的蚂蚁。为了妥善解决这个问题，我们添加了变量 搜索中，它是一个列表，当图首次在当前迭代中向蚂蚁发送信息时，就会将该蚂蚁加入其中。这样，我们就知道这只特定的蚂蚁有权利在需要时接收来自图的信息，因为它实际上正在进行图遍历。图最终会知道蚂蚁何时完成了本次迭代，并在蚂蚁完成时将其从搜索中 列表中移除。因此，如果本次迭代仍有工作要做，这只蚂蚁就能

再次被选中，开始新一轮。每当图将一只蚂蚁加入搜索中 列表时，$maxInfo$ 计数器就会递增。因此，当该计数器达到值 m 时，就不能再向此列表添加蚂蚁（只要有蚂蚁开始或完成任务，该列表就会相应增减）。这样，蚂蚁就不会执行不必要的工作。

在蚁群优化的粗粒度规范中，会使用串行算法的多个副本，并由一个额外的进程对它们进行协调。此处考虑以下几种规范变体：

•无通信的蚁群系统：在无通信蚁群系统的粗粒度规范中，会使用 p 个串行算法实例。这些实例完成后，各自将最优结果发送给辅助进程 B，该进程只需从中选出最优结果。

• $\mathcal{MM}$无通信的蚁群系统：无通信蚁群系统的粗粒度变体与前一种相似。需要指出的是，正如 Manfrin 等人（2006）所述，这是蚁群优化的最佳并行策略。

• 无通信的蚁群系统：此规范与相应的蚁群系统规范非常相似。唯一的区别是必须添加局部更新变换。由于该变换是串行的，因此必须在蚂蚁移动之后引入。

• 带偶尔通信的蚁群系统：在这种蚁群系统粗粒度规范的变体中，串行算法的 p 多个实例同样并行执行，不过这些实例会通过一个通用共享函数，每隔预先设定的若干次迭代共享一次信息素轨迹。该函数可以有不同的定义方式：例如，可以令所有实例复制拥有最优解的算法实例的信息素轨迹；可以令每条边的信息素轨迹都取所有算法实例中该边信息素轨迹的平均值；也可以通过加权平均，使信息素轨迹受到最优算法实例的信息素轨迹的部分影响，等等。这项信息素轨迹更新操作由一个扮演“超图”角色的进程执行。

7 结论与未来工作

本文介绍了若干蚁群优化算法在两种并行化粒度下的进程代数规范；据我们所知，这是首次将这些算法的功能行为与并发行为一并完整地形式化定义。通过构建一种适用于这一目标、结构恰当的进程代数，我们得以检验进程代数能否用于实际规范所讨论领域中的算法。为了使以该语言编写的规范自然易懂，我们在设计时面临若干选择，例如如何协调概率选择与非确定性选择、如何在进程间保存和传递局部信息、这些信息如何影响概率等。所得代数见第3.1节。随后，我们展示了如何使用该代数规范所选领域中的进程，特别是如何规范并行蚁群优化算法。我们介绍了若干类型的实现方法，涵盖了广泛的并行化类型。

通过本研究，我们提出了若干未来研究方向。我们希望将所开发的代数应用于其他群体智能算法的规范。这些算法通常用于解决复杂的大规模问题，因此采用并行方式实现它们已越来越普遍。所提出的代数能够自然地对不同类型的并行化进行建模，因此也适用于这些算法。此外，其中许多算法都需要以与我们在本研究中非常相似的方式引入概率。我们将研究代数需要进行哪些调整（如果确实需要调整的话），以便在不损失具体性的前提下涵盖大量群体智能算法。RFD（河流形成动力学）（Rabanal 等人（2007年））、PSO（粒子群优化）（Poli 等人（2007年））和 ABC（人工蜂群算法）（Kumar、Kumar 和 Jarial（2009年））等算法尤其适合使用 PA^2CO 进行规范，既适用于其顺序形式，也适用于不同类型的并行化形式。

致谢：Mar´ıa Garc´ıa： 方法论、调查、验证、形式化分析、撰写——初稿。**Natalia L´opez：** 概念化、监督、项目管理、撰写——初稿及审阅与编辑。**IsmaelRodr´ıguez：** 概念化、监督、项目管理、撰写——初稿及审阅与编辑。

参考文献

Borisenko, A., Gorlatch, S., 2019年。比较GPU并行元启发式算法与分支定界法在间歇工厂优化中的表现。《超级计算杂志》75，7921–7933。Dorigo, M., Birattari, M., St¨utzle, T., 2006年。蚁群优化。《IEEE计算智能杂志》1，28–39。Dorigo, M., Maniezzo, V., Colorni, A., 1996年。蚁群系统：协作智能体蚁群的优化。《IEEE系统、人类与控制论汇刊B辑》26，29–41。Dorigo, M., St¨utzle, T., 2003年。蚁群优化元启发式算法：算法、应用与进展。施普林格。Eberbach, E., 2007年。用于问题求解的成本演算进程代数：处理困难计算问题的范式转变。《理论计算机科学》383，200–243。Fidanova, S., Atanassov, K., 2008年。带有直觉模糊估计的蚁群优化进程广义网模型，载于：《广义网国际研讨会》，第41–48页。Fidanova, S., Atanassov, K., 2009年。混合蚁群优化进程的广义网模型。《保加利亚科学院院刊》62(3)，1–3。Fidanova, S., Atanassov, K., 2010年。作为蚁群优化算法建模工具的广义网，载于：《大规模科学计算》，《计算机科学讲义》5910，施普林格，第326–333页。Fokkink, W., 2000。《进程代数导论》。施普林格。Fokkink, W., 2013。《进程代数导论》。施普林格科学与商业媒体。Gao, J., Sun, Y., Zhang, B., Chen, Z., Gao, L., Zhang, W., 2019年。基于多GPU的蚁群优化算法并行设计，用于从高光谱图像中提取端元。《传感器》19。

van Glabbeek, R., Smolka, S., Steffen, B., Tofts, C., 1990. 概率过程的反应式、生成式和分层模型，载于：第五届计算机科学逻辑年会 LICS '90 论文集，IEEE计算机学会。第130–141页。Goldberg, D., 1989.《搜索、优化与机器学习中的遗传算法》。Addison-Wesley。Groote, J., Reniers, M., 2002.《通信系统的建模与分析》。MIT出版社。Kennedy, J., Eberhart, R., Shi, Y., 2001. 群体智能。Morgan Kaufmann。Kumar, A., Kumar, D., Jarial, S., 2009. 人工蜂群算法及其在数据聚类中的应用综述。《Cybernetics and Information Technologies》17。Le, D.N., Nguyen, G.N., Garg, H., Huynh, Q.T., Bao, T.N., Tuan, N.N., 2021. 在软件项目管理中使用并行最大最小蚁群系统算法优化多轮采购问题的投标方选择。《Computers, Materials & Continua》66, 993–1010。L´opez, N., N´u˜nez, M., 2004. 概率进程代数及其等价关系概述，载于：《随机系统验证——当前研究指南》，《计算机科学讲义》，2925，施普林格。第89–123页。Loscos, D., Mart´ı-Oliet, N., Rodr´ıguez, I., 2022. 随机局部搜索算法的推广与完备性。《Swarm and Evolutionary Computation》68, 100982。L´opez, N., N´u˜nez, M., 2004. 概率进程代数及其等价关系概述，载于：《随机系统验证》，《计算机科学讲义》，2925，施普林格。第89–123页。Manfrin, M., Birattari, M., St¨utzle, T., Dorigo, M., 2006. 旅行商问题的并行蚁群优化，载于：《蚁群优化与群体智能》，第五届国际研讨会 ANTS 2006，施普林格。第224–234页。Massink, M., Latella, D., 2012. 觅食蚂蚁的流体分析，载于：《协调模型与语言》，施普林格·柏林·海德堡。第152–165页。

de Melo Menezes, B., Kuchen, H., de Lima Neto, F.B., 2022. 群体智能算法的并行化：文献综述。《国际并行编程杂志》50, 486–514。de Melo Menezes, B.A., Herrmann, N., Kuchen, H., Buarque de Lima Neto, F., 2021. 基于算法骨架的高级并行蚁群优化。《Int. J. Parallel Program.》49, 776–801。Milner, R., Parrow, J., Walker, D., 1992. 移动进程演算 I。《信息与计算》100, 1–40。N´u˜nez, M., Rodr´ıguez, I., 2001. PAMR：并发系统资源管理进程代数，载于：网络与分布式系统形式化技术，FORTE 2001，第21届网络与分布式系统形式化技术国际会议，IFIP TC6/WG6.1，Kluwer。第169–184页。Poli, R., Kennedy, J., Blackwell, T., 2007年. 粒子群优化。《Swarm Intelligence》1, 33–57。Rabanal, P., Rodr´ıguez, I., Rubio, F., 2007年. 利用河流形成动力学设计启发式算法，载于：非常规计算国际会议，施普林格。第163–177页。St¨utzle, T., Hoos, H., 2000. 最大最小蚁群系统。《未来一代计算机系统》16, 889–914。Sumpter, D., Blanchard, G., Broomhead, D., 2001. 蚂蚁与智能体：一种蚁群行为建模的进程代数方法。《Bulletin of Mathematical Biology》63。Tofts, C., 1992. 使用进程代数描述社会性昆虫行为。《计算机仿真学会会刊》9, 227–227。