الذكاء الصنعي

وقل رب زدني علما
جامعة دمشق
Damascus University

منشورات جامعة دمشق
كلية الهندسة الميكانيكية والكهربائية
قسم هندسة الحواسيب والأتمتة

الذكاء الصنعي

Artificial Intelligence

تأليف

الأستاذ الدكتور المهندس
سمير كرمان
أستاذ في قسم هندسة الحواسيب والأتمتة

الدكتور المهندس
أسامة أسعد بحبوح
قسم هندسة الحواسيب والأتمتة

جامعة دمشق

1446 – 1447 هـ
2024 – 2025 م

وقل رب زدني علما
جامعة دمشق
Damascus University

الفهرس

المقدمة 9

الفصل الأول

1- مقدمة في الذكاء الصنعي 11

1-1- مقدمة تاريخية 11

1-2- الذكاء الصنعي 13

1-3- خصائص الذكاء الصنعي 37

1-4- الأنظمة الخبيرة 39

الفصل الثاني

2- الوكيل (العميل) الذكي 43

2-1- تعريف الوكيل الذكي 43

2-2- مفاهيم وخواص الوكيل الذكي 44

2-3- دالة الوكيل 47

2-4- الوكيل العقلاني 48

2-5- خواص البيئة المحيطة 51

2-6- بنية الوكلاء الأذكياء 53

2-7- تصنيف الوكلاء حسب برامجهم 54

الفصل الثالث

3- تمثيل المعرفة 61

3-1- مقدمة 61

3-2- تمثيل المعرفة 63

3-3- الشبكات الدلالية 65
3-4- تمثيل المعرفة بالمنطق الرياضي 66
3-5- فئات الكلمات في المنطق الرياضي 68

الفصل الرابع

4- تقنيات البحث 79
4-1- مقدمة 79
4-2- فضاء الحالة 81
4-3- شجرة البحث 82
4-4- الفرق بين فضاء الحالة وشجرة البحث 86
4-5- خوارزمية البحث 86
4-6- البحث الأعمى 91
4-7- البحث التجريبي 98
4-8- مثال 105

الفصل الخامس

5- نظرية الألعاب 107
5-1- مقدمة 107
5-2- البدايات 111
5-3- تعاريف 112
5-4- عناصر اللعبة 115
5-5- أهمية نظرية الألعاب 116
5-6- نظرية الألعاب في الذكاء الصنعي 117
5-7- تصنيف الألعاب 118

5-8- هندسة نظرية الألعاب 132
5-9- أمثلة عن بعض الألعاب 137
5-10- خوارزميات الألعاب 159

الفصل السادس

6- الخوارزميات الجينية 177
6-1- مقدمة 177
6-2- تقنيات البحث 180
6-3- بدايات الخوارزميات الجينية 184
6-4- خلفية بيولوجية 186
6-5- نظرية داروين في التطور 188
6-6- توصيف الخوارزميات الجينية 191
6-7- فضاء البحث 194
6-8- الخطوط العامة للخوارزميات الجينية 196
6-9- ترميز الصبغي 202
6-10- طرائق الترميز 205
6-11- مبادئ وأسس رياضية 217
6-12- خوارزميات الانتخاب 222
6-13- العبور 250
6-14- الطفرة 286
6-15- البرمجة الجينية 305

الفصل السابع

7- لغة البرمجة البرولوغ 337

7-1- مقدمة 337
7-2- تمثيل الحقائق 341
7-3- أنماط المعلنات 344
7-4- بنية المعطيات 348
7-5- بنية البرنامج 350
7-6- مثال توضيحي 357
7-7- أساسيات التعقب الخلفي 368
7-8- مثال متكامل 378
7-9- بعض المعلنات المعرفة مسبقاً في البرولوغ 386
7-10- إجابة الاستعلام 390
7-11- نصائح في كتابة البرامج 394
7-12- مجموعة تمارين 396
7-13- التعامل مع القوائم 399
7-14- التعابير الرياضية 414
7-15- التعقب الخلفي 428
7-16- البرمجة العودية 444
7-17- بعض المسائل الهامة في البرولوغ 454

قائمة المراجع 491

المقدمـــة

علم الذكاء الصنعي ((Artificial Intelligence (AI) هو أحد علوم الحاسب الآلي الحديثة ، التي تبحث عن أساليب متطورة لبرمجته ، للقيام بأعمال واستنتاجات تشابه ولو في حدود ضيقة تلك الأساليب التي تنسب لذكاء الإنسان . فهو بذلك علم يبحث أولاً في تعريف الذكاء الإنساني وتحديد أبعاده، ومن ثم محاكاة بعض خواصه . وهنا يجب توضيح أن هذا العلم لا يهدف إلى مقارنة أو تمثيل العقل البشري بالآلة التي هي من صنع المخلوق، بل يهدف هذا العلم الجديد إلى فهم العمليات الذهنية المعقدة التي يقوم بها العقل البشري في أثناء ممارسته (التفكير)، ومن ثم ترجمة هذه العمليات الذهنية إلى ما يوازيها من عمليات حسابية تزيد من قدرة الحاسب على حل المشاكل المعقدة.

يتسم البشر عن سواهم من المخلوقات الحية بالعقل ، حيث نستخدم قدراتنا العقلية في كل صغيرة وكبيرة في حياتنا . يعنى الذكاء الصنعي بميكنة الذكاء الإنساني ودراسة قدراته العقلية، فمن أهم الأسباب لدراسة الذكاء الصنعي هو محاولة فهمنا لعمليات العقل البشري، بطريقة تبتعد عن علم الفلسفة وعلم النفس وعلم التشريح والتي تعنى بدورها أيضاً بالعقل البشري، فعلم الذكاء الصنعي يكافح لبناء الذكاء بالقدر الذي يعنى فيه بفهم هذا الذكاء.

السبب الثاني لدراسة هذا العلم هو أن برامج الذكاء الصنعي مفيدة في كثير من المجالات في حياتنا التي أصبحت رقمية! فلا يستطيع أحد التنبؤ بأحداث المستقبل، إلا أنه من الواضح أن الحاسوب مع الذكاء الإنساني سيكون له تأثير ضخم وواضح في حياتنا اليومية وفي صناعة الحضارة.

يعد الذكاء الصنعي لغزاً مهماً: فكيف لهذا الدماغ الصغير، سواء أكان بيولوجياً أم إلكترونياً، أن يفهم ويدرك ويتنبأ ويتفاعل مع عالم أكبر وأعقد من الدماغ نفسه؟ كيف لنا أن نسلك طريقاً يعنى بصناعة مثل هذا الدماغ الصغير بكل صفاته المعقدة؟ هذا سؤال صعب، ولكن بخلاف البحث عن وسيلة مواصلات أسرع من سرعة الضوء، فإن الباحث في علم الذكاء الصنعي والدارس له يجد أن هذا العلم قائم على أسس متينة وممكنة، كل ما عليه هو النظر إلى المرآة ليجد مثالاً حياً عن النظام الذكي.

يمكن تعريف الذكاء الصنعي للحاسب الآلي ، بأنه القدرة على تمثيل نماذج حاسوبية (Computer Models) لمجال من مجالات الحياة ، وتحديد العلاقات الأساسية بين عناصره، ومن ثم استحداث ردود الفعل التي تتناسب مع أحداث ومواقف هذا المجال . وبالتالي فالذكاء الصنعي مرتبط أولاً بتمثيل نموذج حاسوبي لمجال من المجالات، ومن ثم استرجاعه وتطويره ، وثانياً بمقارنته مع مواقف وأحداث مجال البحث للخروج باستنتاجات مفيدة. ويتضح أن الفرق بين تعريفي الذكاء الصنعي والإنساني المذكورين أعلاه هو أولاً القدرة على استحداث النموذج، فالإنسان قادر على اختراع وابتكار هذا النموذج، في حين أن النموذج الحاسوبي هو تمثيل لنموذج سبق استحداثه في ذهن الإنسان . وثانياً في أنواع الاستنتاجات التي يمكن استخلاصها من النموذج ، فالإنسان قادر على استعمال أنواع مختلفة من العمليات الذهنية مثل الابتكار (Innovation) والاختراع (Creation) والاستنتاج بأنواعه (Conclusion)، في حين أن العمليات الحاسوبية تقتصر على استنتاجات محدودة طبقاً لبديهيات وقوانين متعارف عليها يتم برمجتها في البرامج نفسها.

الفصل الأول

1- مقدمة في الذكاء الصنعي:

1-1- مقدمة تاريخية:

في منتصف القرن العشرين، بدأ عدد قليل من العلماء استكشاف نهج جديد لبناء آلات ذكية، بناء على الاكتشافات الحديثة في علم الأعصاب، ونظرية رياضية جديدة للمعلومات، وتطور علم التحكم الآلي، وقبل كل ذلك، عن طريق اختراع الحاسوب الرقمي(اختراع آلة يمكنها محاكاة عملية التفكير الحسابي الإنسانية). أسس المجال الحديث لبحوث الذكاء الصنعي في مؤتمر في حرم كلية (Dartmouth) في صيف عام 1956 أصبح هؤلاء الحضور قادة بحوث الذكاء الصنعي لعدة عقود، وخاصة (John McCarthy, Marvin Minsky)، هم وتلاميذهم كتبوا برامجاً أدهشت معظم الناس. كان الحاسب الآلي يحل مسائل في الجبر ويثبت النظريات المنطقية ويتحدث الإنجليزية. بحلول منتصف الستينات أصبحت تلك البحوث تمول بسخاء م ن وزارة الدفاع الأمريكية، ولكنهم فشلوا في إدراك صعوبة بعض المشاكل التي واجهتهم ، وفي بداية السبعينات قطعت الحكومتان البريطانية والأمريكية التمويل. في أوائل الثمانينات، شهدت أبحاث الذكاء الصنعي صحوة جديدة من خلال النجاح التجاري "للنظم الخبيرة"، وهي أحد برامج الذكاء الصنعي التي تحاكي المعرفة والمهارات التحليلية لواحد أو أكثر من الخبراء البشريين. بحلول عام 1985 وصلت أرباح أبحاث الذكاء الصنعي في السوق إلى أكثر من مليار دولار، وبدأت الحكومات التمويل من جديد. في التسعينات وأوائل القرن الواحد والعشرين، حقق الذكاء الصنعي نجاحات أكبر، وإن كان ذلك إلى حد ما وراء الكواليس. يرجع ذلك النجاح إلى عدة عوامل ؛ هي: القوة الكبيرة للحواسيب اليوم، وزيادة التركيز على حل مشاكل فرعية محددة، وخلق

علاقات جديدة بين مجال الذكاء الصنعي وغيرها من مجالات العمل في مشاكل مماثلة، وفوق كل ذلك بدأ الباحثون الالتزام بمناهج رياضية قوية ومعايير علمية صارمة.

الذكاء الصنعي علم معرفي حديث، بدأ رسمياً في الخمسينات من القرن الماضي، أما قبل هذه الفترة، فنجد أن عدداً من العلوم الأخرى عنيت بشكل أو بآخر بالذكاء الصنعي وبطريقة غير مباشرة. باستعراض علم الوراثة، نجد ما يرتبط بالذكاء في حقل دراسة جينات العلماء في محاولة لإعادة سبب ذكائهم للوراثة، وفي مجال الفيزياء نجد أن جميع الطلاب بلا شك يشعروا بأن جميع الأفكار الجيدة أخذت من غاليليو وآينشتاين ونيوتن وبقية العلماء، ولابد من الدراسة لأعوام عديدة حتى يتسنى لأحدهم تقديم اكتشاف جديد!. في المقابل فإن الذكاء الصنعي لا يزال مفتوحاً ليشغِل بدراسته آينشتاين جديد جميع أوقاته.

البحث عن ماهية الذكاء كانت قد شغلت الفلاسفة قبل أكثر من ألفي عام، فقد حاولوا فهم كيف تتم رؤية الأشياء، وكيف يتم التعلم، والتذكر والتعليل. ومع حلول استخدام الحاسوب في الخمسينات تحولت هذه البحوث إلى أنظمة تجريبية واقعية.

حالياً، فإن للذكاء الصنعي تطبيقات عديدة، سواء أكانت تطبيقات ذات أغراض عامة مثل الإدراك والتعليل المنطقي، أم كانت مهمات ذات غرض خاص مثل لعب الشطرنج أو التشخيص الطبي! غالباً فإن الخبراء والعلماء يتوجهون إلى الذكاء الصنعي لحفظ خبراتهم وتجاربهم التي قضوا بها حياتهم. فالذكاء الصنعي مجال عالمي يصلح لجميع التوجهات.

إن الذكاء الصنعي أحدث ما ابتكر العقل البشري في العقود الخمس الأخيرة من القرن العشرين. فقبل وجود الحاسوب أو حتى الالكترونيات والكهرباء

حاول الإنسان إيجاد بعض الأشياء التي تتمتع ببعض صفاته. وعلى سبيل المثال، يذكر أن البابا (Sylvestre II) في القرون الوسطى صنع آلة قادرة على نطق عدد معين من الكلمات والإجابة على بعض الأسئلة بنعم أو لا.

أما العرب فقد برعوا بعلم الحيل ، أي الآلات ذاتية الحركة ، من القرن التاسع للميلاد ، وكانت أبرزها الآلات الموسيقية والساعات والنوافير. ويرى الكثير من ذوي الاختصاصات الهندسية الحيل العربية أنها السلف المباشر للكثير من الأجهزة الحديثة، مثل آلات البيع المحاسبية.

ابتدع العالم الانكليزي (Alan Turing) اختبار للتأكد من ذكاء الآلة ، بحيث كان الاختبار عبارة عن وضع الآلة في حجرة مغلقة تخرج منها نهاية طرفية في ردهة، ووضع إنساناً في حجرة مغلقة أخرى يتصل هو الآخر بنهاية طرفية في الردهة نفسها، ويوجد إنسان آخر هو الحكم في الردهة. وهو الذي يتولى الاتصال بالآلة ويتولى الحكم وأداء الحوار مع كل من الآلة والإنسان لاكتشاف أي الطرفين يتصل بالإنسان دون إن يراهما ويقيس ذكاء الآلة وقدرتها على التفكير ولاقى اختبار (Turing) الكثير من المعارضة لعل أبرزها هو تأثر الاختبار بالحكم. وإن كان قد بدأ يضع الأساس الذي بدأت فيه أبحاث الذكاء الصنعي وذكاء الآلة.

1-2- الذكاء الصنعي:

لا توجد نظرية موحدة أو نموذج يوجه بحوث الذكاء الصنعي. اختلف الباحثون حول العديد من القضايا. من أكثر المسائل التي ظلت دون إجابة لمدة طويلة هي: هل ينبغي للذكاء الصنعي محاكاة الذكاء الطبيعي من خلال دراسة علم النفس أو علم الأعصاب؟ هل يمكن إعادة إنتاج الذكاء باستخدام رموز رفيعة المستوى، على غرار الكلمات والأفكار؟ أم أنها تحتاج إلى معالجة " شبه رمزية"؟

1-2-1- تعريف الذكاء الصنعي:

هو أحد علوم الحاسب الآلي الحديثة، التي تبحث عن أساليب متطورة للقيام بأعمال واستنتاجات تشابه ولو في حدود ضيقة تلك الأسباب التي تنسب لذكاء الإنسان.

هو سلوك وخاصيات معينة تتسم بها البرامج الحاسوبية تجعلها تحاكي القدرات الذهنية البشرية وأنماط عملها. من أهم هذه الخاصيات القدرة على التعلم والاستنتاج ورد الفعل على أوضاع لم تبرمج في الآلة. إلا أن هذا المصطلح إشكالي نظراً لعدم توفر تعريف محدد للذكاء، كما تعرف الكثير من الكتب الذكاء الصنعي على أنه " دراسة وتصميم العملاء الأذكياء "، والعميل الذكي هو نظام يستوعب بيئته ويتخذ المواقف التي تزيد من فرصته في النجاح في تحقيق مهمته أو مهمة فريقه. (John McCarthy)، الذي صاغ هذا المصطلح في عام 1956، عرفه بأنه "علم وهندسة صنع آلات ذكيه".

الذكاء مصطلح يتضمن عادة الكثير من القدرات العقلية المتعلقة بالقدرة على التحليل، والتخطيط، وحل المشاكل (Problem Solving)، وسرعة المحاكات العقلية، كما يشمل القدرة على التفكير المجرد، وجمع وتنسيق الأفكار، والتقاط اللغات، وسرعة التعلم.

ومع أن المفهوم العام السائد عند الناس للذكاء يشمل جميع هذه الأمور ، وربما يجعلها الناس مرتبطة بقوة الذاكرة (Memory)، إلا أن علم النفس يدرس الذكاء كميزة سلوكية مستقلة عن الإبداع (Creativity)، والشخصية (Character)، والحكمة (Wisdom)، وحتى قوة الحافظة المتعلقة بالذاكرة .

والذكاء الصنعي هو أحد العلوم المتفرعة عن علم الحاسوب، وهو العلم المعني بجعل الحواسيب تقوم بمهام مشابهة وبشكل تقريبي لعمليات الذكاء البشرية كالتعلم، والاستنباط، واتخاذ القرارات.

وهناك تعريفات أخرى وردت في الكتب والمراجع ، تتعلق بعلم الذكاء الصنعي، ومن أبرزها قاموس الموسوعة العربية للكمبيوتر والإنترنت الذي عرفه بأنه مصطلح يطلق على علم من أحدث علوم الحاسب الآلي، وينتمي هذا العلم إلى الجيل الحديث من أجيال الحاسب الآلي، ويهدف إلى أن يقوم الحاسب بمحاكاة عمليات الذكاء التي تتم داخل العقل البشري، بحيث تصبح لدى الحاسوب المقدرة على حل المشكلات واتخاذ القرارات بأسلوب منطقي ومرتب وبطريقة تفكير العقل البشري نفسها.

وهذه العمليات تتضمن:

- التعليم: اكتساب المعلومات والقواعد التي تستخدم هذه المعلومات.
- التعليل: استخدام القواعد السابقة للوصول إلى استنتاجات تقريبية أو ثابتة.
- التصحيح التلقائي أو الذاتي.

فالذكاء الصنعي: هو فرع من فروع علوم الحاسوب يُعنى بميكنة السلوك الذكي عند الإنسان. وفيه نحتاج إلى:

- نظام بيانات: يستخدم لتمثيل المعلومات والمعرفة.
- خوارزميات: نحتاج إليها لرسم طريقة استخدام هذه المعلومات.
- لغة برمجة: تستخدم لتمثيل كلاً من المعلومات والخوارزميات.

فعلم الذكاء الصنعي هو أحد علوم الحاسب الآلي الحديثة ، التي تبحث عن أساليب متطورة لبرمجته، للقيام بأعمال واستنتاجات تشابه ولو في حدود ضيقة تلك الأساليب التي تنسب لذكاء الإنسان، فهو بذلك علم يبحث أولاً في تعريف الذكاء الإنساني وتحديد أبعاده، ومن ثم محاكاة بعض خواصه. وهنا يجب توضيح أن هذا العلم لا يهدف إلى مقارنة أو مشابهة العقل البشري الذي خلقه الله جلت قدرته وعظمته بالآلة التي هي من صنع المخلوق، بل يهدف هذا العلم الجديد إلى فهم العمليات الذهنية المعقدة التي يقوم بها العقل البشري في أثناء ممارسته التفكير، ومن ثم ترجمة هذه العمليات الذهنية إلى ما يوازيها من عمليات محاسبية تزيد من قدرة الحاسب على حل المشاكل المعقدة.

1-2-2- فلسفة الذكاء الصنعي:

يطرح الذكاء الصنعي عدة أسئلة، مثل: هل هناك حدود لمدى ذكاء الآلات؟ هل هناك فرق جوهري بين الذكاء البشري والذكاء الصنعي؟ وهل يمكن أن يكون للآلة عقل ووعي؟.

وفي هذا المجال نجد آراء متناقضة جداً للعلماء:

أطروحة (Dartmouth): " يمكن وصف كل جانب من عملية التعلم أو غيرها من مظاهر الذكاء بدقة شديدة ، تمكن الإنسان من تصميم آلة تحاكيه ". وهذا هو موقف معظم الباحثين في هذا المجال.

نظرية عدم الاكتمال: " لا يمكن لنظام منطقي كالبرنامج الحاسوبي، إثبات جميع الجمل الصحيحة ". يعتقد بعض العلماء بأن هذه نظرية وضعت حدوداً لما يمكن أن تفعله الآلات، حيث وضعت حداً لما يمكن استنتاجه حسابياً، ولكنها لم تضع حدوداً لما يمكن أن يفعله الإنسان.

1-2-3- الذكاء الإنساني:

هو جميع العمليات الذهنية من نبوغ وابتكار و تحكم في الحركة والحواس والعواطف.

أما في نطاق دراسة علم الذكاء الصنعي للحاسبات الآلية فيمكن تعريفه في نطاق قدرة الإنسان على تطوير الأشياء وتحليل خواصها والخروج باستنتاجات. فهو بذلك يمثل قدرة الإنسان على تطوير نموذج ذهني لمجال من مجالات الحياة وتحديد عناصره واستخلاص العلاقات الموجودة بينها ، ومن ثم استحد اث ردود الفعل التي تتناسب مع أحداث ومواقف هذا المجال.

من أهم فوائد هذا النموذج الذهني الذي يستحدثه الإنسان لا شعورياً أنه يساعد على حصر الحقائق ذات العلاقة بالموضوع في مجال البحث ، وتبسيط الخطوات المعقدة التي تتميز بها الصورة الحقيقية.

الفرق بين الذكاء الصنعي والذكاء الإنساني:

الذكاء الصنعي: يمكن تعريف الذكاء الصنعي للحاسب الآلي بأنه القدرة على تمثيل نماذج حاسوبية (Computer Models) لمجال من مجالات الحياة وتحديد العلاقات الأساسية بين عناصره، ومن ثم استحداث ردود الفعل التي تتناسب مع أحداث ومواقف هذا المجال . فالذكاء الصنعي بالتالي مرتبط أولاً بتمثيل نموذج حاسوبي لمجال من المجالات، ومن ثم استرجاعه وتطويره. ومرتبط ثانياً بمقارنته مع مواقف وأحداث مجال البحث، للخروج باستنتاجات مفيدة. ويتضح أن الفرق بين تعريفي الذكاء الصنعي والإنساني المذكورين أعلاه هو أولاً القدرة على استحداث النموذج، فالإنسان قادر على اختراع وابتكار هذا النموذج، في حين أن النموذج الحاسوبي هو تمثيل لنموذج سبق استحداثه في ذهن الإنسان . وثانياً في أنواع الاستنتاجات التي يمكن استخلاصها من النموذج ، فالإنسان قادر على

استعمال أنواع مختلفة من العمليات الذهنية مثل الابتكار (Innovation)، والاختراع (Creativity)، والاستنتاج (Reasoning) بأنواعه ، في حين أن العمليات الحاسوبية تقتصر على استنتاجات محدودة طبقاً لبديهيات وقوانين متعارف عليها يتم برمجتها في البرامج نفسها.

الذكاء الطبيعي:

- يرتبط بالقدرة على حل المعضلة.
- التعليم جزء لا يتجزأ من الذكاء.
- الفطرة أو الغريزة التي تكون موجودة، وهي جزء من تشكيلة الذكاء.

بالنسبة للذكاء الصنعي:

- نرى برنامج يقوم بجميع المهام السابقة.
- ليس شرط وجود الحاسب لوجود الذكاء الصنعي.
- يقوم الحاسب في حال وجوده بتسريع عملية الحساب ، وهو الأداة التي تسرع من الوصول إلى الحل.

مثال:

مركبة الفضاء كاسينة:

عبارة عن مركبة جزء منها ميكانيكي ، وجزء برمجي قام به المهندسون الذين كان لديهم تصور من لحظة إرسال الروبوت وحتى لحظة إرسال الصور من الفضاء.

لا نهتم في الجزء الميكانيكي أو الجزء الالكتروني ، وإنما التحكم أي إيجاد الخوارزمية التي ستأخذ القرار باتجاه دوران المحرك.

1-2-4- هدف الذكاء الصنعي:

الهدف من الذكاء الصنعي بعيد المدى ومن أين أتى الذكاء الصنعي؟

الهدف: ماذا سنفعل في الذكاء الصنعي والطرائق المطبقة لصناعة الآلات ذات السلوك الذكي

لماذا استخدم الحاسوب؟

لأنه يعمل أشياء مفيدة وأفضل لنا ، ويجري حسابات كبيرة وسريعة. يوجد حواسيب تعتمد على الحساب وهي سريعة ، وأخرى تعتمد على الاستنباط وهي ذكية. هدفنا عمل حواسيب تقوم بالأمور نفسها التي يقوم بها الإنسان ، ولكن بذكاء.

يهدف الذكاء الصنعي إلى قيام الحاسوب بمحاكاة عمليات الذكاء التي تتم داخل العقل البشري ، بحيث تصبح لدى الحاسوب المقدرة على حل المشكلات واتخاذ القرارات بأسلوب منطقي ومرتب بطريقة تفكير العقل البشري نفسها. وتمثيل البرامج الحاسوبية لمجال من مجالات الحياة ، وتحسين العلاقة الأساسية بين عناصره.

1-2-5- أساليب الذكاء الصنعي:

يتركز أصل علم الذكاء الصنعي في أبحاث بحتة ونظرية تدرس أساليب تمثيل النماذج في ذاكرة الحاسب الآلي (Model Representation) وطرائق البحث والتطابق بين عناصرها (Match Methods & Search) واختزال أهداف بها (Goal Reduction) وإجراء أنواع الاستنتاجات المختلفة (Reasoning) مثل الاستنتاج عن طريق المنطق (Logic) أو عن طريق المقارنة (Analogy) أو عن طريق الاستقراء (Induction).

سوف نعرض فيما يأتي لأهم هذه الأساليب:

1- أسلوب استخدام القوانين:

استخدام القوانين (Rules) التي تحكم مجالاً من المجالات هو من أهم أساليب تمثيل هذه النماذج، فلو كانت أنواع الفاكهة مثلاً هي مجال بحثنا ، فإنه يمكننا كتابة القانون الآتي: إذا كان النبات فاكهة وكان لونها أحمر فهي غالباً تفاح، ويحتوي هذا القانون على قسمين:

القسم الشرطي (Premise) المتمثل في " إذا كان النبات فاكهة وكان لونها أحمر ".

القسم الاستنتاجي أو الفعلي (Action) المتمثل في " فهي غالباً تفاح ".

وباستخدام عدد كبير من هذه القوانين عن موضوع معين فإننا ننشئ نموذجاً ضمنياً يخزن الحقائق عن موضوع البحث، ويمكن استخدامه في التعامل مع الأحداث والخروج باستنتاجات عن موضوع البحث، ويعتبر هذا النوع من التمثيل من الأساليب الشائعة نظراً لسهولة تطبيقه إلا أنه يعتبر تمثيلاً بسيطاً ولكن يعجز في كثير من الأحيان عن تمثيل جميع أنواع النماذج واستخراج جميع أنواع الاستنتاجات المعروفة.

2- أسلوب شبكات المعاني:

يعد أسلوب شبكات المعاني (Semantic Networks) من الأساليب الشائعة في تمثيل النماذج ، وهو يتلخص في إنشاء شبكة من العلاقات بين عناصر النموذج.

3- أسلوب تمثيل الإطارات:

هو من أساليب التمثيل الشائعة (Frame Representation)، والذي يمكن اعتباره نوعاً خاصاً من تمثيل شبكات المعاني.

4- أسلوب الرؤية الحاسوبية:

يتلخص أسلوب الرؤية الحاسوبية (Computer Vision) في تحويل الصورة الرقمية المكونة من نقاط (Pixels) سوداء أو بيضاء إلى خطوط وأضلاع متصلة لتكوين صورة . ثم مقارنة خصائص الصورة الناتجة بالنماذج المخزونة سابقاً في الجهاز. يمكن بهذه الطريقة التعرف مثلاً على صورة الطائرة من أجنحتها وذيلها، وتمييز المطار بمدرجات إقلاع الطائرات، والمسجد من مئذنته ، وهكذا. وتتمثل صعوبة الرؤية الحاسوبية في اختلاف الصورة مع اختلاف الإضاءة المسلطة على الجسم ووقوع الظل على أجزاء منه، ولتقنية الرؤية الحاسوبية تطبيقات عديدة في مجالات توجيه الصواريخ والطائرات والتوابع (الأقمار الصناعية) ومجالات التجسس، بالإضافة طبعاً لمجال الأذرع الآلية.

ومن أشهر الأنظمة التي تستعمل الرؤية الحاسوبية في المجال الصناعي هو نظام (Onsight)، المستخدم الآن في شركة جنرال موتورز للسيارات بكندا ، والذي يسمح للذراع الآلية الذكية، بفرز قوالب محركات السيارة (Engine Casts) في أثناء مرورها أمامه على الحزام المتحرك تحت إضاءة معينة. وبعد تحليل الضوء تقوم الذراع باستخراج القوالب التي لا تتفق والمواصفات المطلوبة.

يمثل استعمال أكثر من ذراع واحدة في حيز ضيق صعوبة فنية كبيرة نظراً لخطورة اصطدام بعضها ببعض، كما أن التنسيق بينها في التعاون على إنجاز عمل ما له مشاكله الفنية ، نظراً لضرورة متابعة كل ذراع وما يقوم به من عمل بالإضافة إلى ما أنجز غيره من أعمال. وقد اقتصر استعمال الأذرع الآلية إلى عهد قريب على استخدام كل ذراع على حدة، حيث أن استخدام أكثر من ذراع واحدة في إنجاز مهمة مركبة يحتاج إلى أنظمة آلية جديدة ومعقدة تقوم برسم الخطة العامة للحركة وتقوم باستنتاج الخطوات المنطقية التي يجب أن تنفذها كل

ذراع. وبالتالي فهي أنظمة تحتاج إلى الذكاء الصنعي وأساليبه في استحداث نماذج حاسوبية للبيئة ، وتخزين قوانين وأسس الحركة المطلوبة . ورغم ظهور بعض الأنظمة الآلية تمكن الذراع الآلية من الحركة الذاتية مثل نظام " Strips " إلا أن معظم هذه الأنظمة ما زال في طور البحث والتطوير.

5- أسلوب معالجة اللغات الطبيعية (Natural Language Processing):

يسعى هذا الأسلوب إلى فهم اللغات الطبيعية بهدف تلقين الحاسوب الأوامر مباشرة بهذه اللغة ، وبالتالي تمكين الحاسوب من المحادثة مع الناس عن طريق الإجابة عن أسئلة معينة، ويتضمن هذا الأسلوب ما يأتي:

- **الكلام** (Speech):

تزويد الحاسوب بمعلومات وبرامج حتى يكون لديه القدرة على فهم الكلام البشري عن طريق تلقي الأصوات من الخارج وإعادة تجميعها والتعرف عليها ومن ثم الرد عليها.

- **النظر** (Vision):

تزويد الحاسوب بأجهزة استشعار ضوئية تمكنه من التعرف على الأشخاص أو الأشكال الموجودة.

- **الروبوتية** (Robotics):

وهو آلة كهروميكانيكية تتلقى الأوامر من حاسوب تابع لها، فيقوم بأعمال معينة، والذكاء الصنعي في هذا المجال يشتمل على إعطاء الروبوت القدرة على الحركة وفهم محيطه والاستجابة لعدد من العوامل الخارجية.

- **التعليم** (Learning):

أهمها التعليم المعزز آلياً، وهو محاولة الاستفادة من طاقات الحاسوب في مجالات التربية والتعليم.

1-2-6- مشاكل الذكاء الصنعي:

انقسمت مشكلة محاكاة (أو خلق) الذكاء إلى عدد من المشاكل الفرعية المحددة. وتتكون هذه من سمات أو قدرات معينة يود الباحثون أن يجسدوها نظاماً ذكياً، كالاستنتاج، والتفكير المنطقي، والمقدرة على حل المشكلات.

وضع الباحثون الأوائل في علم الذكاء الصنعي الخوارزميات التي تحاك ي التفكير المنطقي المتسلسل الذي يقوم به البشر عند حل الألغاز، ولعب الطاولة أو الاستنتاجات المنطقية. وفى ثمانينيات وتسعينيات القرن الماضي ، أدت أبحاث الذكاء الصنعي إلى التوصل لوسائل ناجحة للتعامل مع المعلومات غير المؤكدة أو غير الكاملة، مستخدمة في ذلك مفاهيم من الاحتمالية والاقتصاد.

بالنسبة للمشاكل الصعبة، تتطلب معظم هذه الخوارزميات موارد حسابية هائلة مما يؤدي إلى " انفجار اندماجي "، أي يصبح مقدار الذاكرة أو الوقت اللازم للحواسيب فلكي عندما تتجاوز المشكلة حجماً معيناً. فأصبح البحث عن خوارزميات أكثر قدرة على حل المشكلات هو أولوية قصوى لأبحاث الذكاء الصنعي.

في القرن الواحد والعشرين، أصبحت أبحاث الذكاء الصنعي على درجة عالية من التخصص والتقنية، وانقسمت إلى مجالات فرعية مستقلة بشكل عميق. نمت أقسام المجال حول مؤسسات معينة، وعمل الباحثون على حل مشكلات محددة، وخلافات في الرأي نشأت منذ زمن طويل حول الطريقة التي ينبغي أن يعمل وفقاً لها الذكاء الصنعي، وتطبيق أدوات مختلفة على نطاق واسع.

1-2-7- مجالات الذكاء الصنعي:

نتج من معامل أبحاث الذكاء الصنعي تقنيات عديدة مازال بعضها في الأطوار الأولى من الدراسة والبحث، في حين وصل البعض الآخر إلى نضج نسبي أدى إلى تطوير أنظمة جديدة عملية تعالج مشاكل واقعية كان يعتقد من المستحيل معالجتها بأساليب البرمجة التقليدية، ويعتبر مجال ي الذراع الآلية الذكية (Smart Robot) والأنظمة الخبيرة (Expert Systems) أهم مجالين من هذه المجالات، وفيما يأتي نبذة مبسطة لهاتين التقنيتين وإمكاناتهما:

1- الذراع الآلية الذكية:

استخدمت الذراع الآلية مؤخراً في المصانع للقيام بالأعمال الروتينية التي تحتاج إلى قوة عضلية، ولا تتطلب عمليات أو أنشطة ذهنية معقدة ، مثل عمليات اللحام والدهان في مصانع السيارات. وقد اعتمد تشغيل هذه الأذرع على دقة وسرعة أنظمة التحكم (Control Systems) التي تعمل بوساطة أجهزة الحاسب الآلي . كان اليابانيون أول من استعمل هذه الأذرع بصورة موسعة في صناعة السيارات، والذي نتج عنه غزو اليابان للأسواق العالمية بسيارات ذات جودة عالية وأسعار منافسة.

لاستخدام الأذرع الآلية في التصنيع فوائد عديدة ، فهي لا تطالب بإجازات أسبوعية أو سنوية أو عرضية، ولا تكل ولا تتعب من العمل ، ولا تتوقف إلا خلال فترات الصيانة. كما أنها تستطيع العمل في مصانع غير مكيفة أو مضاءة إضاءة قوية، وفي هذا توفير للطاقة. كما أنها لا ترفع الدعاوى، ولا تطالب بتعويضات إذا تعرضت، خطأ أو عمداً إلى غازات سامة أو مواد كيماوية ضارة مثلاً. وأخيراً فهي لا تحتاج إلى مرافق مساندة مثل دور الحضانة وصالات الطعام والصالات الرياضية وغيرها مما يطالب به العمال . وليس من الصعب طبعاً ترجمة كل هذه

المزايا إلى توفير كبير في تكلفة الإنتاج ، وفي السيطرة على الطاقة الإنتاجية للمصانع، بحيث تتناسب مع قوى العرض والطلب للسوق، وذلك دون اللجوء إلى تسريح العمال لبضعة أسابيع أو شهور، أو في وضع ورديات إضافية.

ومع تطور أنظمة التحكم الآلية وازدياد قدرة الحاسبات الآلية التي تشغلها ، ازدادت قدرات الذراع الآلية ، وأصبحت تقوم بأعمال دقيقة ومركبة ، كصنع شرائح المعالجات، وغيرها من الأعمال التي تتطلب أنظمة تحكم معقدة وصعبة . إلا أن هذه الأعمال كانت محدودة بما يمكن إنجازه باستخدام أساليب البرمجة التقليدية ، وقد أدى إدخال أساليب الذكاء الصنعي في برمجة هذه الأذرع إلى فتح آفاق جديدة لم تكن ممكنة من قبل، فأصبحنا اليوم نتكلم عن أذرع تستعمل الرؤية الحاسوبية في فرز المنتجات ، وفي تحريك الذراع (أو عدة أذرع) في حيز ضيق ، بأسلوب مرن يتناسب مع متغيرات البيئة التي يعمل بها.

2- الأنظمة الخبيرة:

هي برامج حاسوبية، تحتوي على كمية هائلة من المعلومات التي يملكها خبير إنساني في حقل معين من حقول المعرفة، وبعض هذه البرامج أثبتت فعاليتها ودقتها . فللنظام الخبير هو برنامج مصمم لينفذ مهاماً متعلقة بالخبرة البشرية . يحاول النظام الخبير القيام بعمليات تعد عادة من اختصاص البشر ، ويتضمن الحكم واتخاذ القرارات.

يملك الخبراء البشريون كمية هائلة من المعرفة المتخصصة في مجالات عملهم، لذا فإن النظم الخبيرة تستند عادة إلى قواعد معرفة ، وتتضمن عدداً هائلاً من قواعد المعطيات التي تحوي معلومات المعرفة، و تعد النظم الخبيرة في بعض المراجع فرع من الذكاء الصنعي.

ويتكون النظام الخبير من ثلاثة أجزاء رئيسة:

- قاعدة معرفة (Knowledge Base): تتضمن المعارف المتعلقة بحقل الخبرة.
- محرك الاستدلال (Inference Engine): نظام لمعالجة المعارف واستنتاج طريقة الاستدلال.
- واجهة المستخدم (User Interface): تمكن المستخدم غير الخبير من الوصول إلى معرفة النظام الخبير.

ولفظ الخبير مشتق من الخبرة، وهو الشخص المتمرس الذي مر بتجارب عديدة صقلت فهمه لمجال من المجالات، وأغنت فكره بمعلومات اختص بها دون غيره، وميزته عن أنداده من المختصين في المجال ، وبذلك استحق لفظ خبير. وتهدف الأنظمة الخبيرة إلى تطوير برامج حاسوبية تستطيع تحليل الأحداث والمواقف في مجال من المجالات ، والوصول إلى الاستنتاجات أو النتائج التي يصل لها الخبير.

ويتم ذلك عن طريق استحداث نموذج حاسوبي يوازي النموذج الذهني الذي لدى الخبير وخزن المعلومات داخله. وقد دلت الأبحاث على أن المعلومات التي يستخدمها الخبير في عمله تنقسم إلى قسمين رئيسين: الأول خاص بالمعلومات الشائعة في هذا المجال مثل الحقائق (Facts) والقوانين (Rules) المتعارف عليها والمقبولة لجميع المختصين (Heuristics) التي يتميز بها الخبير عن غيره ، والتي قد تكون على شكل علاقة مثلاً بين لون البشرة ونسبة الكوليسترول في الدم. أو الشكل الانسيابي لعينة صخرية ونسبة الترسبات المعدنية فيها.

وهذه القوانين يستخلصها الخبير من التجارب التي مر بها ، وتقوم بتوجيه بحثه ودراسته للحالة المعروضة عليه ، ومساعدته في الوصول إلى النتائج المطلوبة. وقد تختلف هذه القوانين التخصصية من خبير إلى آخر.

كانت الورقة العلمية التي تقدم بها البروفيسور فايجنباوم (faygenbaum) خبير الذكاء الصنعي في جامعة ستانفورد لمؤتمر الذكاء الصنعي العالمي لعام 1977 م أكبر الأثر في توجيه هذا العلم الجديد، فقد طرح البروفيسور فكرة أن قوة الأنظمة الخبيرة تنبع من المعرفة التي تختزنها ، وليس من قدرتها على تمثيل النماذج والقيام بعمليات استنتاجية. ومن هذه النظرية ركزت الأبحاث الجديدة على استخلاص المعرفة من الخبراء، عوضاً عن التركيز على الطرائق المختلفة للتمثيل والعمليات الاستنتاجية المعقدة، وهما موضوعان لم يتم تكوين نظريات متكاملة عنهما بعد، وبالتالي فهما يعانيان من قصور في تطبيقاتهما العملية.

ومن أوائل الأنظمة الخبيرة التي تطورت حتى الآن نظام (Mycin) لتحليل وعلاج أمراض الدم المعدية . وقد طور هذا النظام في جامعة ستانفورد حيث احتوت قاعدة معلوماته على نحو (400) قانون تربط العوارض المحتملة للمرض بالاستنتاجات الممكنة. وقد قورنت النتائج المستخرجة من نظام (Mycin) في كثير من تحليلاته على مستوى الأطباء الموجودين في اللجنة.

كما يعد نظام (Prospector) أيضاً، من أنجح الأنظمة التي طورت حتى الآن، حيث قام باكتشاف ترسبات معدن الموليبدنم (Molybdenum–) في ولاية واشنطن بالولايات المتحدة في الأماكن التي قرر الخبراء عدم جدوى البحث فيها . وقد بلغت قيمة هذا الاكتشاف نحو مائة مليون دولار أمريكي.

ومجال الأنظمة الخبيرة هو حديث الساعة في مجال الذكاء الصنعي، وذلك نظراً لكونها أنجح التطبيقات العملية لهذا العلم الجديد، وتوجد اليوم شركات

عدة تسوق ما يسمى بقشرة أو هيكل النظام (Expert Shells)، وهي أنظمة تسهل عملية تمثيل النماذج الحاسوبية، وتخزين قوانينها ، ومن ثم إجراء الاستنتاجات عنها بصورة آلية، وبذلك يتم التركيز على استخلاص المعرفة من الخبير أو الخبراء ووضعها في قوانين (Rules) تناسب وأسلوب عمل هيكل النظام المختار، وتسمى هذه العملية بهندسة المعرفة (Knowledge Engineering). كما يسمى الذين يقومون بها مهندسي المعرفة (Knowledge Engineers). ويوجد حالياً في الأسواق هياكل أنظمة خبيرة عديدة تختلف في نقاط تفوقها وضعفها وفي أسعارها ومجالات تطبيقها . كما ظهرت أخيراً هياكل أنظمة تعمل على الحاسب الشخصي وبأسعار مقبولة نسبياً ، مما يشير إلى قرب وصول هذه الأنظمة إلى الأسواق التجارية بأسعار منافسة.

ورغم النجاح الذي حققته كثير من هذه الأنظمة ، فإنه يجب توخي الحذر وعدم التسليم لكل ما يخرج من هذه الأنظمة من نتائج أو استنتاجات . كما يجب الابتعاد عن الخوض في توقعات خيالية عن قدراتها. والذي يجب توضيحه هو أن هذه الأنظمة لا يمكن أن تحل محل الخبير نهائياً، وأنه على الرغم من أن كثيراً من النتائج التي تتوصل لها الأنظمة تتطابق أو حتى تفوق النتائج التي قد يصل لها الخبير ، إلا أن هذه الأنظمة تستخلص قوتها من التركيز على موضوع معين ومحدود لمجال من المجالات، وأنه كلما اتسع نطاق هذا الموضوع، ضعفت قدرتها الاستنتاجية ، والعكس صحيح. و بذلك فإن الأنظمة الخبيرة ذات فائدة كبيرة ما دامت تستخدم من قبل شخص مختص بموضوع مجال البحث ومطلع على الأساليب والتحاليل التي يستخدمها النظام في الوصول إلى استنتاجاته، وهي مفيدة في يد أنصاف الخبراء ذوي المعرفة الجديدة بالموضوع ، إلا أنها قد تؤدي إلى نتائج عكسية . فمثلاً إذا وضع نظام مثل نظام (Reactor) الذي يحلل أخطاء

المفاعلات النووية، في يد شخص لا يعرف عن المفاعلات النووية شيئاً ، وتصور هذا الشخص أنه بذلك أصبح خبيراً، وبدأ يعبث بالمفاتيح، فإن النتائج سوف تكون خطيرة بلا شك.

وللأنظمة الخبيرة مجالات معينة أثبتت قدرتها فيه أكثر من غيرها ، فقد اشتهرت في التخطيط (Planning) وفي تحليل العوارض وتحديد الأخطاء (Diagnostics) وفي التصميم (Design) وفي القيادة والسيطرة (Command and Control)، وغيرها من المجالات المتخصصة التي تم فهم العمليات المطلوبة لها، والتي تتناسب والقدرات التمثيلية والاستنتاجية لهياكل الأنظمة المستخدمة. نستنتج من كل ما تقد م أن الأنظمة الخبيرة أو بالأحرى نظم قواعد المعرفة (Knowledge Base Systems) كما يفضل كثير من الباحثين تسميتها ، هي أنظمة جديدة ذات قدرات تفوق بمراحل قدرات الأنظمة الآلية التقليدية، حيث أن لها القدرة في الحصول على الاستنتاجات بمعلومات متناقضة وغير مكتملة (Incomplete and Inconsistent knowledge). وهي بذلك تحاكي الخبراء والقادة العسكريين ، الذين غالباً ما يتخذون القرارات تحت هذه الظروف، وهي تقنية عملية مفيدة مادامت تستخدم من قبل المختصين وطبقت في المجالات التي تتناسب مع حدود معرفتنا لقدراتها.

يتفوق النظام الخبير على البرامج الحاسوبية التقليدية، حيث يختلف النظام الخبير عن البرامج الاعتيادية في الحاسوب، في أن المعرفة وثيقة الصلة بموضوع معين، وأساليب الاستفادة من هذه المعرفة مندمجة مع بعض . يبدو نموذج حل المشكلة في النظام الخبير كقاعدة معرفة قائمة بذاتها بدلاً من أن يكون جزءاً من البرنامج العام. وبهذا يكون بإمكان النظام الخبير إدخال بيانات إلى قاعدة المعرفة المتوافرة دون الحاجة إلى إعادة البرمجة. وبهذا يمكننا القول إن برنامج الحاسوب

التقليدي ينظم المعرفة بمستويين هما البيانات وقاعدة المعرفة، والسيطرة. ومن هنا نجد أن الاختلاف بين الأنظمة الحاسوبية الخبيرة وبرامج الحاسوب التقليدية في حل المسائل التي ليست لها طريقة حل مسبقة يتجلى في ما يأتي:

- كونها تعمل بالرموز بدلاً من الأرقام ، وبهذا تفتح مجالات جديدة لمعالجتها بوساطة الحاسوب.
- الاستدلال (Reasoning) وطريقة البحث التقنية (Heuristics).
- كونها تتعامل مع اللغات المبنية على المفسر (Interpreter) وليس المترجم (Compiler). حيث تسمح بالتعامل مع ا لتعابير المبنية على المفاهيم الصعبة في اللغات التقليدية ، والتعبير عن المشكلة بلغة الذكاء الصنعي مثل (PROLOG, LISP)، والتي تتحول إلى إجراءات خلال التنفيذ ، وبهذا لا يكون على المبرمج أن يعرف مسبقاً الحل أو النتيجة.

يتبين من هذا ، أنه ليس كل نظام يستند إلى قاعدة المعرفة هو نظام خبير، ولكن أن يمتلك القدرة على التفسير والوصول إلى القرارات، وطلب معلومات إضافية، كما يفعل الإنسان الخبير في عملية التفسير والتحليل والتحري ، وخاصة في المجالات التي تكون فيها الحقائق ناقصة أو غير أكيدة.

1-2-8- أهمية استخدام الذكاء الصنعي:

ظهر الذكاء الصنعي في الآونة الأخيرة من القرن الماضي ومطلع هذا القرن، واستخدم في التحكم الصناعي والتطوير الطبي وإيجاد الحلول المثلى والتحقيق الأمني والجنائي. ولعل ازدياد الصعوبات وتعقيدها وعدم فائدة الحلول

البرمجية المتواضعة قاد العلماء والمهتمين في فهم آلية التفكير البشري وكيفية معالجة المعلومات وتخزينها واسترجاعها وذلك بالاعتماد على أسلوب المحاكاة (Simulation) في حل هذه المعضلات . وقد تم التوصل إلى هيكلية برمجية مترابطة ، مكونة من أوامر برمجية ومصفوفات رياضية وجبر بولي اني تسمى الشبكات العصبية الصنعية، وهو جعل الآلة تتصرف بذكاء نيابة عن الإنسان بكل فاعلية ومرونة. وقد أثبتت البحوث والتجارب نجاحها بنسب عالية . ولكي تقوم هذه الشبكة بعملها تحتاج إلى فكرتين هما:

1- فكرة الإشراف: وتعني وجود شخص يقدم للآلة أمثلة لمرة واحدة فقط ، تقوم بعد ذلك بحفظها واسترجاعها عند الحاجة إليها.

2- التعلم بدون مشرف: إذ تقوم هذه الفكرة على تقديم عدد من النماذج المتشابهة، والتي على أساسها تميز أي نموذج جديد يقدم إليها.

ولتبسيط فكرة الذكاء الصنعي نقدم المثال الآتي:

بفرض أن شخصاً مريضاً يعاني من حالة مرضية وعند إجراء التحاليل والفحوصات صرف له دواء لهذا المرض. في هذه الحالة نقوم بإدخال نوع المرض و أسبابه وأعراضه وطرائق الوقاية منه ونوع الدواء المستخدم للقضاء عليه إلى الحاسب الآلي. بعد فترة زمنية، جاء مريض يعاني من المرض نفسه، فما علينا إلا إدخال التحاليل والفحوصات إلى الحاسوب ليتم مقارنتها مع الحالة السابقة ، فإن تمت المطابقة ، صرف العلاج حتى في عدم وجود الطبيب ، لأن الآلة هنا قامت بعمل الطبيب.

1-2-9- تطبيقات علم الذكاء الصنعي:

تطبيقات الذكاء الصنعي كثيرة جداً ومن أكثرها شيوعاً:

1-2-9-1- استخدام الذكاء الصنعي في المكتبات ومراكز المعلومات:

هناك إجماع في الرأي ، بأن الذكاء الصنعي سيكون تكنولوجيا جديدة ، يبحث فيها المتخصصون في مجال المكتبات والمعلومات عن الطرائق المفيدة لاستخدامها واستثمارها ، لتسهيل أعمالهم وتحسين نوعية خدماتهم وخبراتهم الخاصة. وقد استغل المتخصصون هذه التكنولوجيا، وقاموا بإنتاج العديد من النظم في الخزن والاسترجاع وفي الفهرسة والاستخلاص والأعمال المرجعية .

فالمتخصصون يجب أن تتوافر لديهم الخبرة، والتفاعل مع مظاهر الحياة المختلفة ومهارات أخرى مثل التصنيف ، والخبرة الأكاديمية ، وإجراء المقابلات ، والمعرفة باحتياجات المستفيدين.

ومن نماذج الأنظمة المستخدمة في المكتبات:

1- نموذج (Coder): وهو مشروع طور من قبل (Fox)، غرضه تطوير قاعدة معرفة تشمل تحليل الوثائق واسترجاعها ويتألف من فرعين:

- نظام فرعي تحليلي ، يتعلق بإدخال ومعالجة وتمثيل الوثائق الجديدة.
- نظام فرعي استرجاعي، يسمح باسترجاع وثيقة أو جزء منها.

2- نموذج (Rebeic): نظام يبحث في أنماط الكلمات ضمن نصوص البحث الآلي المباشر ، بدلاً من استرجاع وثائق مكشفة مسبقاً قاعدة المعرفة اعتمدت على (Ruies) وصعوبته كونه يوفر قواعد متخصصة لكل مستفيد.

3- نموذج (Esscape): مشروع تم فيه بناء نظامين خبيرين في فهرسة المكتبة . والعمل الرئيسي فيه هو اختبار نقاط وصول لتحديد المداخل الرئيسة والإضافية ، والاستنتاج هو إمكانية استخدام النظام في الفهرسة لإنتاج القيود الببلوغرافية الصحيحة، ويكون مفيداً أيضاً في الأعمال غير التقليدية.

4- (Gemi): هو نظام خبير تم تطبيقه في مجال استرجاع المعلومات ، وهو مبني على القواعد، حيث يمكن المستفيد من معرفة المرجع في مجال اهتمامه مع توفير ببلوغرافية ومستخلص لجميع المراجع المتوافرة في المكتبات الجامعية. طبق هذا النظام في العراق في حقل المكتبات والمعلومات ، وقد تم الأخذ بعين الاعتبار عند تطبيق النظام طبيعة المستفيد ومستواه الثقافي والمهني.

1-2-9-2- ألعاب الحاسوب:

ويتم في هذه الألعاب وضع مشكلة أمام الفرد ويحاول حل تلك المشكلة. وبعض هذه الألعاب تكون صعبة للغاية بحيث أن الفرد العادي لا يستطيع التوصل إلى حلولها . وبالتالي فقد وضع مصممو تلك البرامج مستويات يستطيع الفرد تحديد المستوى الذي يستطيع اجتيازه بنجاح . وبعض هذه الألعاب تكون متدرجة، ويبدأ الفرد فيها بالمستوى السهل ثم المتوسط ثم العالي. فباستخدام الذكاء الصنعي، أصبح الحاسوب نداً قد يصعب التغلب عليه في كثير من الألعاب.

1-2-9-3- الأنظمة الخبيرة:

وهي نظم حاسوبية معقدة تقوم على تجميع معلومات متخصصة من الخبراء البشريين، ووضعها في صورة تمكن الحاسوب من تطبيق تلك المعلومات أو الخبرات على مشكلات مماثلة.

1-2-9-4- معالجة اللغة البشرية:

وهو ما يختص بتطوير برامج و نظم لها القدرة على فهم أو توليد اللغة البشرية . أي أن مستخدم هذه البرامج يقوم بإدخال البيانات بصورة طبيعية والحاسوب يقوم بفهمها والاستخلاص منها.

1-2-9-5- التعلم الآلي:

وهو جعل الحاسوب يتعلم كيفية حل المشاكل بنفسه ، إما بالتعلم من اكتساب الخبرات السابقة أو من خلال تحليل الحلول الصحيحة واستنباط طريقة الحل منها أو حتى من التعلم من خلال الأمثلة.

1-2-9-6- معالجة اللغات الطبيعية:

معالجة اللغات الطبيعية (Natural Language Processing) هي علم فرعي من علوم الذكاء الصنعي والتي بدورها متفرعة من المعلوماتية، وتتداخل بشكل كبير مع علوم اللغويات التي تقدم التوصيف اللغوي المطلوب للحاسوب، هذا العلم يمكننا من صناعة برمجيات تتمكن من تحليل ومحاكاة فهم اللغات الطبيعية.

1-2-9-7- تحليل النصوص الطبيعية:

أولى الأنظمة مثل (SHRDLU)، التي عملت في بيئة محددة من الكلمات، عملت بشكل فعال، ما قاد الباحثين إلى التفاؤل الشديد الذي تلاشى بسرعة عندما تم تطبيق الأنظمة في بيئات أكثر واقعية بوجود التعقيد والإبهام (عدم الوضوح) في اللغات التي يتداولها البشر.

1-2-9-8- فهم اللغات الطبيعية:

يشار إليه أحياناً بمشكلة الذكاء الصنعي الكاملة، لأن تمييز وفهم اللغات الطبيعية يحتاج إلى معرفة مكثفة بالعالم الخارجي والقدرة على التحكم به.

تعريف "الفهم " هو واحد من المشاكل الرئيسة في معالجة اللغات الطبيعية.

مثال على بعض المشاكل التي تواجه أنظمة فهم وتحليل اللغات الطبيعية:

جملة "أعطينا القردة الموزة لأنها كانت جائعة" ، وجملة "أعطينا القردة الموزة لأنها كانت ناضجة" ، لهما ذات التكوين القواعدي، ولكن الضمير "ها" في كلمة "لأنها" تعود في الأولى على القردة، وفي الثاني ة على الموزة . ففهم الجملة بشكل صحيح غير ممكن دون معرفة خصائص الموز وسلوك القردة.

- مستويات تحليل اللغات الطبيعية:

بالنسبة للنصوص المكتوبة، فإن تحلي لها يمر في عدة مراحل تختلف باختلاف طريقة التحليل ، ولكن إحدى أكثر أساليب التحليل انتشاراً هو تتبع المراحل الآتية:

- التحليل الصرفـي: وهو الجزء الذي يهتم بمعرفة نوع الكلمات، واحتوائها على الضمائر، وغيرها من المعلومات الصرفية.
- التحليل النحوي: وهو الجزء الذي يهتم بعلاقة الكلمات بعضها مع بعض، وهيكلية الجملة، وغيرها من المعلومات النحوية، ويعتمد على المرحلة الصرفية.
- التحليل الدلالي: وهو الجزء الذي يهتم بفهم المقصود من الجملة ، عن طريق الربط المنطقي بين ما يدور الحديث عنه في الجملة وبين العالم الواقعي، ويعتمد على كل من المرحلة الصرفية والنحوية.
- المجالات الرئيسية لمعالجة اللغات الطبيعية:
- القراءة الآلية للنصوص وتنقيحها.
- تمييز الكلام.
- توليد النصوص أو الكلام آلياً.

- الترجمة الآلية وتقنياتها.
- فهم الأسئلة والإجابة عنها.
- إيجاد المعلومات.
- استخلاص المعلومات، والتلخيص الآلي.

1-2-10- مكونات الذكاء الصنعي:

يقوم علم الذكاء الصنعي ككل على مبدأين أساسيين فقط هما:

- تمثيل البيانات:

وهو كيفية تمثيل البيانات أو المشكلة في الحاسوب ، بحيث يتمكن الحاسوب من معالجتها و إعطاء الخرج المناسب . أي كيفية وضع المشكلة في صورة ملائمة للحاسوب بحيث يفهمها ويتمكن من التفكير في حل لها. تجدر الإشارة إلى أن هناك لغات تستخدم في عملية تمثيل البيانات منها لغة (OWL) و(RDF) المستخدمتان الآن في تمثيل البيانات في الويب (Semantic Web).

- البحث:

وهو ما نعتبره التفكير بحد ذاته، حيث يقوم الحاسوب بالبحث في الخيارات المتاحة أمامه وتقييمها طبقاً لمعايير موضوعة له، أو قام هو باستنباطها بنفسه ثم يقرر الحل الأمثل.

يتكون الذكاء الصنعي من ثلاثة مكونات أساسية هي:

أ- قاعدة المعرفة (Knowledge Base) غالباً ما يقاس مستوى أداء النظام بدلالة حجم ونوعية قاعدة المعرفة التي يحتويها، وتتضمن قاعدة المعرفة:

1. الحقائق المطلقة: تصف العلاقة المنطقية بين العناصر والمفاهيم ومجموعة الحقائق المستندة إلى الخبرة والممارسة للخبراء في النظام.

2. طرائق حل المشكلات وتقديم الاستشارة .

3. القواعد المستندة على صيغ رياضية.

ب- منظومة آلية الاستدلال (geneinference)، وهي إجراءات مبرمجة تقود إلى الحل المطلوب من خلال ربط القواعد والحقائق المعينة لتكوين خط الاستنباط والاستدلال.

ج- واجهة المستفيد (User Interface) وهي الإجراءات التي تجهز المستفيد بأدوات مناسبة للتفاعل مع النظام خلال مرحلتي التطوير والاستخدام.

1-3- خصائص الذكاء الصنعي:

- التمثيل الرمزي:

تتعامل البرامج مع رموز تعبر عن المعلومات المتوفرة مثل: الجو اليوم حار ، والسيارة خالية من الوقود ، وأحمد في صحة جيدة ، والطعام له رائحة زكية. وهو تمثيل يقترب من شكل تمثيل الإنسان لمعلوماته في حياته اليومية.

- البحث التجريبي:

تتوجه برامج الذكاء الصنعي نحو مشاكل لا تتوافر لها حلول يمكن إيجادها تبعاً لخطوات منطقية محددة. إذ يتبع فيها أسلوب البحث التجريبي كما هو حال الطبيب الذي يقوم بتشخيص المرض للمريض، فأمام هذا الطبيب عدد من الاحتمالات قبل التوصل إلى التشخيص الدقيق، ولن يتمكن بمجرد رؤيته للمريض وسماع آهاته من الوصول إلى الحل . وينطبق الحال على لاعب الشطرنج، فـإن حساب الخطوة التالية يتم بعد التفكير باحتمالات وافتراضات متعددة. وهذا الأسلوب من البحث التجريبي يحتاج إلى ضرورة توافر سعة تخزين

كبيرة في الحاسوب ، كما تعتبر سرعة الحاسوب من العوامل الهامة لدراسة الاحتمالات الكثيرة ودراستها.

– احتضان المعرفة وتمثيلها:

لما كان من الخصائص الهامة في برامج الذكاء الصنعي استخدام أسلوب التمثيل الرمزي في التعبير عن المعلومات، وإتباع طرائق البحث التجريبي في إيجاد الحلول فإن برامج الذكاء الصنعي يجب أن تمتلك في بنائها قاعدة كبيرة من المعرفة تحتوي على الربط بين الحالات والنتائج، مثل:

- إذا كان مشغل الأقراص في جهاز الكمبيوتر لا يقرأ البيانات المسجلة على القرص ، والقرص جيد ، ومشغل القرص سليم ، والكابل سليم، فإن العطل يكون في مشغل الأقراص نفسه.
- إذا كان الجو غير صحو، ودرجة الحرارة منخفضة ، فيجب ارتداء المعطف.

وفي هذه الأمثلة يتضح التمثيل الرمزي (الجو غير صحو)، واحتضان المعرفة بمعرفة عطل مشغل الأقراص وبمعرفة وجوب ارتداء العطف.

– البيانات غير المؤكدة أو غير المكتملة:

يجب على البرامج التي تصمم في مجال الذكاء الصنعي أن تتمكن من إعطاء حلول إذا كانت البيانات غير مؤكدة أو غير مكتملة. وليس معنى ذلك أن تقوم بإعطاء حلول مهما كانت الحلول خاطئة أم صحيحة . وإنما يجب لكي تقوم بأدائها الجيد ، أن تكون قادرة على إعطاء الحلول المقبولة ، وإلا تصبح قاصرة. ففي البرامج الطبية إذا ما عرضت حالة من الحالات دون الحصول على نتائج التحليلات الطبية فيجب أن يحتوي البرنامج على القدرة على إعطاء الحلول.

- القدرة على التعلم:

تعد القدرة على التعلم إحدى مميزات السلوك الذكي، وسواء كان التعلم في البشر يتم عن طريق الملاحظة أم الاستفادة من أخطاء الماضي فإن برامج الذكاء الصنعي يجب أن تعتمد على استراتيجيات لتعلم الآلة.

1-4- الأنظمة الخبيرة:

الأنظمة الخبيرة هي أنظمة صنع قرار ، أو أي أجهزه حاسوبية وبرمجيات لحل المشاكل، وتستطيع أن تصل إلى مستوى معين من الأداء تساوي أو حتى تتعدى الخبراء البشريين في بعض الاختصاصات.

إن الأنظمة الخبيرة بطبيعتها هي فرع تطبيقي من الذكاء الصنعي وهناك عدة تطبيقات على الأنظمة الخبيرة، في التشخيص الطبي ، واستكشاف المعادن، وتكوينات الحاسوب. كما أن الأنظمة الخبيرة تنتشر في مجالات تطبيقية معقدة كإدارة العقارات، وخطط الشركات، وتحليل الخطأ.

الفكرة الأساسية وراء الأنظمة الخبيرة بسيطة، فالخبرة تنتقل من الخبراء إلى الحاسوب ، ويستدعيها مستخدمو الحاسوب كنصيحة معينة عند الحاجة ، ويستطيع الحاسوب أن يتوصل إلى استنتاجات معينة، وبعد ذلك تنصح الأنظمة الخبيرة الشخص المحتاج إلى الاستشارة لاتخاذ القرار المناسب.

تستخدم الأنظمة الخبيرة الآن في الآلاف من المنظمات وتخدم العديد من المهام. هذه الإمكانات تزود الشركات بإنتاجية محسنة وميزات تنافسية هائلة.

1-4-1- أهمية الأنظمة الخبيرة :

تأتي أهمية هذا النوع من البرامج من خلال قدرته على استخلاص الخبرات الإنسانية وتخزينها حاسوبياً بتقليد الخبير في عمله بالمستوى نفسه.

والأهمية الأكبر عندما تبدأ الدول بمعرفة ضرورة نقل هذه الخبرات من خلال البرامج على أسطوانات صغيرة، وليس من خلال الاستثمار البشري المكلف. تعد الأنظمة الخبيرة أحد تطبيقات علم الذكاء الصنعي الذي يهدف إلى نقل الذكاء البشرى إلى نظم الحاسبات، عن طريق تصميم البرمجيات وأجهزة الحاسبات التي تحاكى سلوك وتفكير البشر.

1-4-2- خواص الأنظمة الخبيرة:

- تستخدم أسلوب مشابه للأسلوب البشري في حل المشكلات المعقدة
- تتعامل مع الفرضيات بشكل متزامن وبدقة وسرعة عالية.
- وجود حل متخصص لكل مشكلة ولكل فئة متجانسة من المشاكل.
- تعمل بمستوى علمي واستشاري ثابت لا يتذبذب.
- يتطلب بناؤها تمثيل كميات هائلة من المعارف الخاصة بمجال معين.
- تعالج البيانات الرمزية غير الرقمية من خلال عمليات التحليل والمقارنة المنطقية.

1-4-3- دوافع استخدام الأنظمة الخبيرة:

- لأنها تهدف لمحاكاة الإنسان فكراً وأسلوباً.
- لإثارة أفكار جديدة تؤدي إلى الابتكار.
- لتخليد الخبرة البشرية.

- توفير أكثر من نسخة من النظام تعوض عن الخبراء.
- غياب الشعور بالتعب والملل.
- تقليص الاعتماد على الخبراء البشر.

1-4-4- عنصرا النظام الخبير:

- أهل الخبرة: وهم الأفراد الذين يقومون بإعداد الأنظمة وإدخالها في الحواسيب ومعالجة الخلل في حال حدوثه.
- المستفيدون من النظام: وهم المدراء أو الأشخاص الذين يستعينون بالنظام للبحث عن حلول لمشكلة معينة.

1-4-5- مزايا النظام الخبير:

- يحتفظ النظام الخبير بمعارف متراكمة ، ويجعلها جاهزة على الفور.
- يساعد الموظفين الجدد وحديثي العهد بالمهنة في بلوغ مستويات عالية من الإنتاجية في وقت قصير.
- سهلة الاستخدام بوساطة غير المتخصصين.

وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الثاني

2- الوكيل (العميل) الذكي (Intelligent Agents):

2-1- تعريف الوكيل الذكي:

عبارة عن كائن يستطيع إدراك بيئته (Environment) التي يكون موجوداً فيها، وذلك عبر المستشعرات (Sensors) التي يمتلكها هذا الكائن ومن ثم التجاوب معها بوساطة آليات التنفيذ (Actuators) أو الجوارح.

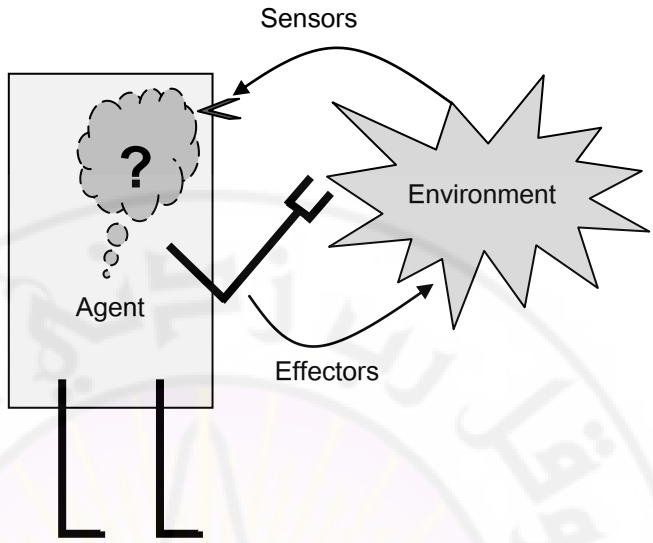
كما يعرف بأنه كينونة قادر على استقبال المعلومات من الوسط المحيط به من خلال الحساسات ، وقادر على التأثير بالوسط المحيط بوساطة أعضاء تأثيره.

ومن التعاريف الأخرى، أنه كينونة برمجية مخصصة لإنجاز مهمة محدّدة لصالح مستخدم أو لصالح وكيل آخر ، ومن أجل ذلك يستخدم المعرفة أو تمثيل أهداف ورغبات المستخدم أي توصيف للمعضلة.

من وجهة نظر برمجية ودون التطرق إلى برنامج الوكيل المنطقي، فهو وكيل ذكي لمعرفة أي عمل نختار، وبالتالي نستنتج أن الوكيل الذي أبحث عنه هو التأثير المنطقي والأسباب هي أنه احتاج إلى وكيل ذكي يفهم السلوك.

يمكن أن نعد الإنسان وكيلاً، فهو يملك أدوات استشعار (الحواس الخمس) وبها يستطيع إدراك بيئته، كما أن لديه جوارح (الأيدي والأرجل) يؤثر بها في بيئته. كذلك الروبوت لديه أدوات استشعار مثل كاميرا التصوير، ومقياس درجة الحرارة.

يمكن اعتبار أي برنامج كوكيل،لأنه يستطيع تحسس وإدراك البيئة عن طريق لوحة المفاتيح والفأرة، وتبدو ردة الفعل وآثار عمله واضحة على الشاشة، كما في الشكل(2-1).



الشكل(2-1) الوكيل الذكي

الوكيل	أعضاء التأثير	الحساسات
إنسان	اليدان، الذراعان	العينان، الأذن
روبوت	أذرع روبوتية	كاميرا، حساسات، أشعة
برنامج	معطيات، سلاسل محارف	معطيات، سلسلة محارف

2-2- مفاهيم وخواص الوكيل الذكي:

تعد مفاهيم الوكيل من الأساسيات في فهم عمله؛ وهي:

- الإدراك (Percept): يقصد به البيانات التي يتلقاها الوكيل عن طريق مستشعرات الدخل (Input).
- ردة الفعل (Action): الأحداث الصادرة عن الوكيل.

- دالة الوكيل (Agent Function): تكون على شكل جدول أو صيغة رياضية.
- برنامج الوكيل (Agent Program): هو الذي يمثل سلوك الوكيل.

كما يتمتع الوكيل بخواص، منها:

تفاعل مترو (يأخذ وقتاً)	تفاعل انعكاسي
تواصلي	انعزالي
صعب (لا يقبل الأخذ والرد)	مرن (أحسن)
قابل للتعليم	مبرمج – قابل للبرمجة
مضاد	متعاون – متنافس
غير منطقي	منطقي
متحكم به	تلقائي
متحرك	ثابت
استنساخ ذاتي	قابل للتطور

ويوجد خواص أخرى تدرس في (Multi Agent).

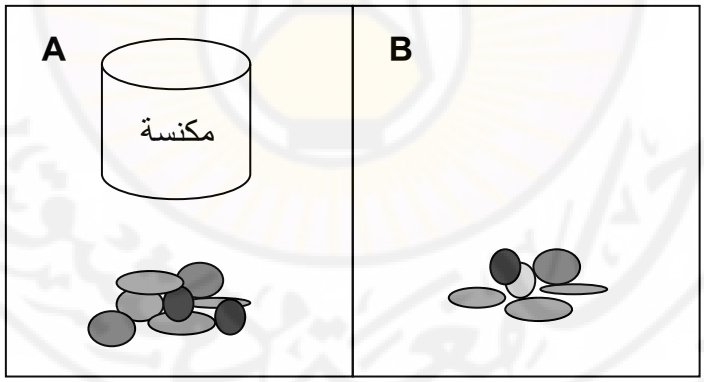
مثال (1):

مكنسة التنظيف (Vacuum Cleaning): هنا مثال لمكنسة كهربائية تقوم بتنظيف مربعين (A, B)، كما في الشكل(2-2)، وتقع داخل إحداهما في كل لحظة.

إن الأحداث الواردة، أي ما يدركه الوكيل هو المربع الذي يقع بداخله (A) أو (B).

أما حالة المربع من ناحية مستوى النظافة فهي نظيف (Clean) أو غير نظيف (Dirty).

كما أن ردود الفعل الصادرة (Action) هي إما الاتجاه يميناً (Right) أو الاتجاه يساراً (Left) أو القيام بعملية الشفط (Suck).



الشكل(2-2) مثال للوكيل الذكي

مثال (2):

وكيل منّظم (Rhino Robot): يقود الناس من منطقة إلى أخرى، ويتكلم عن الآثار وعن المنطقة عندما يصل إلى المنطقة المطلوبة.

مثال (3):

الوكلاء المتحركون: يجب على الروبوت الاستمرار في رؤية الهدف وأخذ القرار السريع والسليم، علماً أنه لا يعرف توضع الحواجز ، ومسار الهدف غير معروف بشكل مسبق. عندما يكون الوسط ساكناً (Static)، فإنه إذا تحرك الهدف احتاج إلى وسيط للرصد، وبالتالي يجب أن أبحث عن هذا الوسط، إما عن طريق الرؤية أو شيء آخر . أما عندما يكون الوسط ديناميكياً (Dynamic)، فعندها تصبح المسألة أكثر تعقيداً.

2-3- دالة الوكيل (Agent Function):

يمكن استنتاج دالة الوكيل لمثال المكنسة بالشكل الآتي:

إذا كان المربع الحالي نظيفاً ، انتقل إلى المربع الآخر، فيما عدا ذلك قم بالشفط. ونكتب الجدول المبين:

Percept Sequence	Action
(A, Clean)	Right
(A, Dirty)	Suck
(B, Clean)	Left
(B, Dirty)	Suck
(A, Clean), (A, Clean)	Right
(A, Clean), (A, Dirty)	Suck
:	:
(A, Clean), (A, Clean), (A, Clean)	Right
(A, Clean), (A, Clean), (A, Dirty)	Suck
:	:

2-4- الوكيل العقلاني (Rational Agent):

الوكيل العقلاني أو الوكيل المنطقي، هو الوكيل الذي يعتمد على التفكير ويتصرف بشكل صحيح أو بسلوك سليم، أي الوكيل الذي يقوم بتنفيذ أشياء صحيحة. وهذا يعني رياضياً أن كل صف من صفوف جدول دالة (Agent function) تحتوي على بيانات صحيحة. كما يوصف بأنه الوكيل الذي تعطي استجابته أكبر قيمة ممكنة لمعيار الأداء إذا أخذ بعين الاعتبار معلوماته السابقة عن البيئة، وكذلك معلوماته الحالية القادمة من المستشعرات.

وهنا نحتاج إلى أن نقرر أداء الوكيل حتى نعرف إذا كان الوكيل منطقي أم لا. والمنطقية ليست مباشرة بالحكم على أدائه، أي يجب أن أرى أكثر من تنفيذ حتى أستطيع أن أقرر المنطقية أم لا.

يسعى الوكيل المنطقي إلى انجاز الشيء الصحيح معتمداً على ما يدركه وعلى ما يستطيع إنجازه من الإجراءات. أما الوكيل المنطقي المثالي فيكون التفعيل بين الإجراءات المختارة ، حتى يرفع من أدائه اعتماداً على البديهيات والمعرفة الداخلية التي يملكها الوكيل.

إن ما هو منطقي في وقت ما يتعلق بعدة أمور كقياس الأداء، ومعرفة الوسط المحيط والإجراءات وسلاسل الإدراك. وأيضاً يعتمد قياس الأداء على تنفيذ المهمة، وعلى كمية المواد المستهلكة لتنفيذ المهمة ، وعلى الزمن اللازم من أجل تنفيذ المهمة . ويكون قياس الأداء إما داخلياً بوجود إمكانيات لقياس الأداء عن طريق البرنامج ، أو خارجياً بوجود راصد للأداء . وتنفذ عميلة القياس بأشكال مختلفة كالقياس المستمر أو القياس الدوري. يمكن عد نظام السيارة المؤتمتة ذاتية القيادة مثالاً عن الوكيل المنطقي ، الذي يقوم بالانتقال من مدينة لأخرى بقيادة سليمة ودون التعرض لحوادث سير، مع اختيار أقصر طريق للوصول.

أبسط وكيل هو نظام التحكم (PID)، ثم نتجه باتجاه العملاء الأكثر ذكاءً، وهم العملاء القادرين على القيام بعملية التفكير أي عملية أخذ القرار عموماً.

أما الوكيل التلقائي (Autonomous Agent) فهو نظام معلوماتي حسابي يقوم بشكل تلقائي مستقل باستقبال المعلومات من الوسط المحيط مهما بلغت ديناميكيته وتعقيده ، ويؤثر بوسطه من أجل تحقيق المهام التي صمم من أجلها. وقد يكون الوكيل تلقائياً كاملاً، وهو معقد التحقيق أو ليس تلقائياً كاملاً.

إن العوامل التي تحدد عقلانية الوكيل هي:

1) قيمة معيار الأداء (Performance Measure Value):
يجب أن نلاحظ أن الوكيل يقدم مجموعة من ردود الفعل بعد استشعاره للبيئة وهذه الردود تؤثر في البيئة المحيطة، إن مقدار تطابق حالة البيئة مع ما يتوقعه الوكيل يحدد فعالية أداء الوكيل.
2) معلومات الوكيل السابقة عن البيئة المحيطة (Environment).
3) ردود فعل الوكيل وتجاوبه مع التغييرات الحادثة في البيئة ، أو بشكل آخر آليات التنفيذ (Actuator).
4) سلسلة أحداث الإدراك التي تسجلها أدوات الاستشعار (Sensor).

أمثلة على تحديد علاقة الوكيل بالبيئة:

الوكيل	معايير الأداء	البيئة	آليات التنفيذ	المستشعرات
نظام تسوق كتب عبر الإنترنت	الوصول للكتب التي تهمني بأقل	الإنترنت	آلية الإظهار للمستخدم، والروابط،	صفحات النت، وطلبات المستخدم

	التكاليف		وحقول الصفحات	
كرة قدم الروبوت	الربح، والأهداف	الملعب، والفريق، والخصم، والكرة	وسيلة التحرك واالتصدي والرفس	الكاميرا، وحساسات التلامس، والعجلات، وحساسات التسارع
نظام تشخيص طبي	الشفاء بأقل التكاليف	المريض، والمشفى	أسئلة، وتحاليل، ومعالجة	أعراض، واستنتاجات، وإجابات المريض

يؤدي الوكيل ثلاث وظائف:

1. استقبال ظروف المحيط.
2. تفسير المعرفة المستقبلية بالتفكير وحل المعضلة الموافقة حسب نوع الوكيل.
3. تقرير إجراءات الموافقة حسب نوع الوكيل حيث يكون لديه إمكانات بسيطة أو معقدة.

تتحدد بنية الوكيل الذكي بالبرنامج والهيكلية أو بواحد منهما، فالبرنامج هو البروتوكول الذي يوافق بين سلاسل الإدراك والإجراءات . أما الهيكلية فتقوم بالتنظيم المادي والبرمجي، والتحكم بالحساسات و إنتاج سلاسل الإدراك ، وتنفيذ البرامج والإجراءات والتحكم بالمؤشرات.

من أجل أي وكيل يجب تحديد العوامل (PEAS)، وهي اختصار للمعايير (Performance Measure, Environment, Actions, Sensors)، أي قياس الأداء، والوسط ، والإجراءات، و الحساسات . فبالنسبة للسيارة ذاتية القيادة يكون الأداء هو السرعة، والأمن. أما الوسط فهو الطرقات، والمستخدمين الآخرين للطرقات، والزبائن. أما الإجراءات فهي العجلات التي تنتج الفعل كاتخاذ الاتجاه المناسب. وتبقى الحساسات التي تكون عبارة عن كاميرا، وحساسات فوق صوتية، وجهاز (GPS)، وغيرها.

2-5- خواص البيئة المحيطة:

1. واضحة وضبابية (Fully, Partially Observable):

نقول عن بيئة الوكيل أنها واضحة أو مرصودة، إذا كانت المستشعرات تعطي الوكيل المعلومات التي يطلبها، أي الوكيل يملك تصوراً واضحاً عن بيئته، كحل الكلمات المتقاطعة. والعكس صحيح إذا كانت المعلومات المتوافرة جزئية، فنقول أن البيئة ضبابية كقيادة السيارة.

2. محددة واحتمالية (Deterministic, Stochastic): نقول عن بيئة ما أنها محددة، إذا كنا نعرف الحال الذي ستؤول إليه البيئة (Next State) انطلاقاً من الوضع الراهن (Current State)، ومن ردة فعل الوكيل ، كحل الكلمات المتقاطعة. ونقول أنها احتمالية، إذا كانت معرفتنا غير مؤكدة بل وفق قيمة احتمالية، كلعبة النرد أو قيادة المركبة . طبعاً، إذا كنا نعرف كل الحالات ، ماعدا سلوك الوكلاء الآخرين ، فإننا نقول أن البيئة إستراتيجية (Strategic)، كلعبة الشطرنج، والتي يكون فيها وكيلين بالوسط نفسه.

3. متتالية وغير متتالية (Sequential, Episodic): نقول عن بيئة ما أنها متتالية، إذا كانت ردود فعل الوكيل فيها تعتمد على ردود فعله السابقة، كلعبة الشطرنج وقيادة السيارة. ونقول أنها غير متتالية أو متقطعة، إذا لم تكن ردود الفعل تعتمد على ردود الفعل السابقة ، كعملية فحص المعلبات آلياً، وهي أسهل لأن الوكيل لا يحتاج إلى التفكير في المستقبل.

4. ساكنة وديناميكية (Static, Dynamic): نقول عن بيئة أنها ساكنة ، إذا لم تكن متغيرة في أثناء عملية اتخاذ القرار ، كلعبة الشطرنج. ونقول أنها ديناميكية أو متحركة، إذا كانت متغيرة في أثناء عملية اتخاذ القرار ، كعملية قيادة السيارات، أو تحرك الروبوت ضمن صالة بيع يتحرك فيها الزبائن. ويوجد الوسط نصف الساكن وهو الوسط الذي تتغير نقاط الأداء فيه مع الزمن ، كلعبة الشطرنج مع ساعة توقيت.

5. مستمرة ومتقطعة (Continues, Discrete): نقول عن بيئة أنها مستمرة، إذا كان التغير من حالة إلى أخرى يتم في وقت متصل ، كقيادة السيارة. ونقول أنها متقطعة ، إذا كان التغير من حالة إلى أخرى يتم في زمن متقطع ، أو إذا كانت الإجراءات المنفذة من قبل الوكيل منتهية، كلعبة الشطرنج ولعب الورق.

6. أحادية الوكلاء ومتعددة الوكلاء (Single Agents, Multi Agents): نقول عن البيئة أنها أحادية الوكيل، إذا كان هناك وكيل واحد يؤثر في البيئة ، كبرنامج التشخيص المرضي. بينما نقول أن البيئة متعددة الوكلاء ، إذا كان هناك أكثر من وكيل ، وهو حال معظم الألعاب . إن عملية التخاطب بين الوكلاء تستدعي الكثير من أدوات التواصل الداخلية، وتوجد لغة للتخاطب بين الوكلاء تسمى ((Agent Communication Language (ACL)، ومن أنواعها (Knowledge Query and Manipulation Language

((KQML)، وأيضاً (FIPA). ومن تفاصيل الأفعال التي تتم بهذا النوع من التواصل نذكر التأكيد (Confirm)، والإبلاغ (Inform)، والاقتراح (Propose)، والطلب (Request).

ولا بد من الإشارة إلى أن هناك بيئة متعددة الوكلاء تنافسية (Competitive)، كلعبة الشطرنج . وبيئة متعددة الوكلاء تعاونية (Cooperative)، مثل قيادة السيارات. حيث أن سلامة أي وكيل من سلامة بقية الوكلاء. في بعض الأحيان يكون من كمال عقلانية الوكيل الحفاظ على الاتصال مع الوكلاء (Communication). وأحياناً أخرى تكون صفة البيئة الضبابية هامة حتى يتمكن الوكيل من الهروب من فخ توقع تحركاته.

يوضح الجدول الآتي خواص البيئة:

الوكيل	الوضوح	التحديد	التتالي	السلوك	الاستمرارية	التعددية
السيارة ذاتية القيادة	ضبابية	احتمالية	متتالية	ديناميكية	مستمرة	متعددة
نظام تسوق كتب عبر الإنترنت	ضبابية	محددة	متتالية	ساكنة	متقطعة	أحادية
كرة قدم الروبوت	ضبابية	احتمالية	متتالية	ديناميكية	مستمرة	متعددة
الشطرنج مع ساعة	واضحة	إستراتيجية	متتالية	نصف	متقطعة	متعددة
التشخيص الطبي	ضبابية	احتمالية	متتالية	ديناميكية	مستمرة	أحادية
روبوت جودة صناعي	ضبابية	احتمالية	متقطعة	ديناميكية	مستمرة	أحادية
الكلمات المتقاطعة	واضحة	محددة	متتالي	ساكنة	متقطعة	أحادية
لعبة النرد	ضبابية	احتمالية	متتالية	ديناميكية	متقطعة	متعددة

2-6- بنية الوكلاء الأذكياء (Agents Structure):

يوصف الوكيل ببنيته الداخلية أو هيكليته وبآلية عمله، فكل وكيل عبارة عن معمارية وبرنامج. فالمعمارية هي التي تختص بتنظيم العناصر المادية

والبرمجية ضمن الوكيل، بالإضافة إلى التحكم بالمستشعرات وإنتاج سلاسل الإدراك وبالتالي تنفيذ ردود الفعل والبرامج والإجراءات، والتحكم بالمفعلات. أما البرنامج فهو البرتوكول الذي يوافق ما بين سلاسل الإدراك وردود الفعل، وبمعنى أوضح، هو الذي يقوم بتنفيذ دالة الوكيل.

بالطبع هناك فرق بين برنامج الوكيل ودالة الوكيل، فمدخلات دالة الوكيل هي سلسلة حوادث الإدراك كاملة. أما مدخلات برنامج الوكيل فهي حدث الإدراك الحالي فقط. مع الإشارة إلى أنه عندما تكون ردة الفعل تعتمد على سلسلة حوادث الإدراك، فإن على الوكيل أن يحتفظ بالسلسلة في مكان آخر غير البرنامج.

2-7- تصنيف الوكلاء حسب برامجهم:

سوف نتعرف على مجموعة من الوكلاء، وهم:

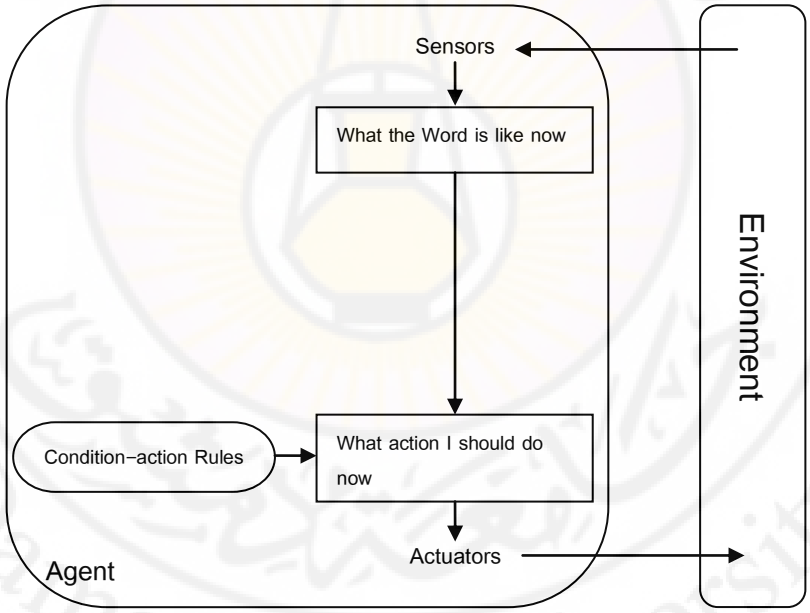
2-7-1- وكيل يعتمد على ردة فعل بسيطة (Simple reflex Agents):

أبسط الأنواع لأنه لا يلجأ إلى حوادث الإدراك كاملة، وإنما الإدراك الحالي فقط، فهو بدون ذاكرة، كما في الشكل(2-3). المبدأ الرئيسي لهذا النوع هو قاعدة (حالة – ردة فعل) (Condition-Action Rule)، حيث نلاحظ وجود الوسط وهو شيء مهم لأنه يعطي الوكيل عن طريق الحساسات المدخلات اللازمة من الوسط المحيط، ثم يختار على ضوء المعلومات ما هو الإجراء الذي يجب القيام به في هذا الوضع اعتماداً على قاعدة (إذا تحقق الشرط ..... نفذ العبارة ...).

فإذا كان الوكيل هو برنامج للتحكم بسيارة، فإنه سيلاحظ إضاءة المصابيح الخلفية للسيارة التي أمامه، وهذا يعني أنها توقفت وبالتالي سيتوقف . يتميز هذا النوع بسهولته، ولكن هذه السهولة هي ثمن لمحدودية قدراته، إذ إنه لا يكون مجدياً إلا إذا كان القرار المتخذ يعتمد بشكل كامل على حدث الإدراك الحالي فقط، أو إذا

كانت بيئة العمل واضحة وضوحاً تاماً ، إذ أن قليلاً من الضبابية يتسبب في مشاكل حقيقة. كما أنه لا يملك ذاكرة فمثلاً إذا نفذ إجراءً باللحظة (t)، فإنه سوف يعود وينفذ الإجراء نفسه باللحظة (t+1)، مثل الروبوت الذي يختار الالتفاف لليمين لوجود طاولة أمامه مثلاً، فإذا أصبح بوضع جديد يعيد العمل السابق حتى ينفذ كامل المهمة الموكلة له. وبالتالي يجب تطويره إلى وكيل مبني على رد الفعل مع ذاكرة.

علماً، أنه أفضل من الوكيل الذي يعتمد على جدول تقابل فقط، ودون إدخال الوسط بعين الاعتبار، فهنا تستخدم قواعد المعرفة لأخذ القرار المناسب.



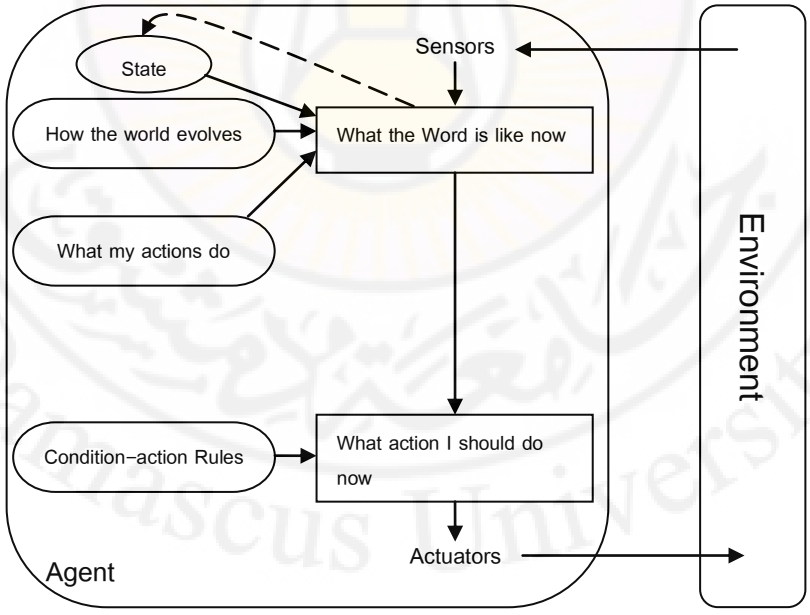
الشكل(2-3) وكيل يعتمد على ردة فعل بسيطة

2-7-2- وكيل يعتمد على نموذج وكيل (Model-Based Reflex):

أحد أكثر الأنواع فعالية في البيئة الضبابية ، وذلك لأن الوكيل يمتلك تصوراً عن حالة بيئته (Internal State)، كما في الشكل(2-4)، ويتم تحديث هذا التصور باستمرار على مرحلتين هما:

1. تحديث المعلومات عن التغييرات التي تحدث في البيئة بدون تدخل من الوكيل نفسه ، أي التغييرات التي لا تعتمد على ردة فعل الوكيل ، مثل، اقتراب السيارات الأخرى من سيارة الوكيل نتيجة زيادة سرعتها أو غروب الشمس.

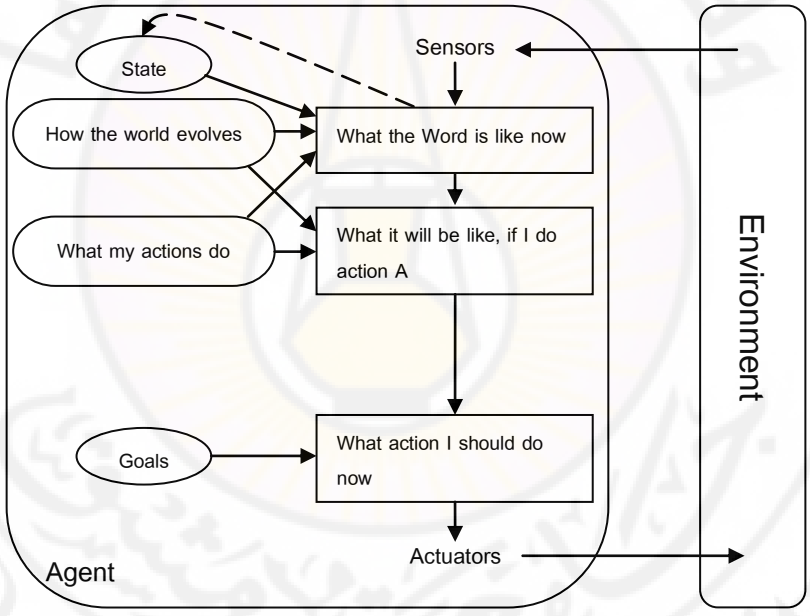
2. تحديث المعلومات عن التأثيرات تحدث في البيئة نتيجة ردود أفعال الوكيل، مثل، وصول الوكيل إلى شمال المنطقة التي كان فيها لأنه انطلق إلى الشمال. وبعد تحديث المعلومات عن العالم الوسيط بوساطة المستشعرات وتفاعل هذه المعلومات مع المعلومات السابقة ، يكون الوكيل تصوراً جديداً عن العالم المحيط وبموجب هذا التصور الجديد يحدد الوكيل ردة الفعل المناسبة.



الشكل(2-4) وكيل يعتمد على نموذج وكيل

2-7-3- وكيل ذو هدف معين (Goal-Based Agents):

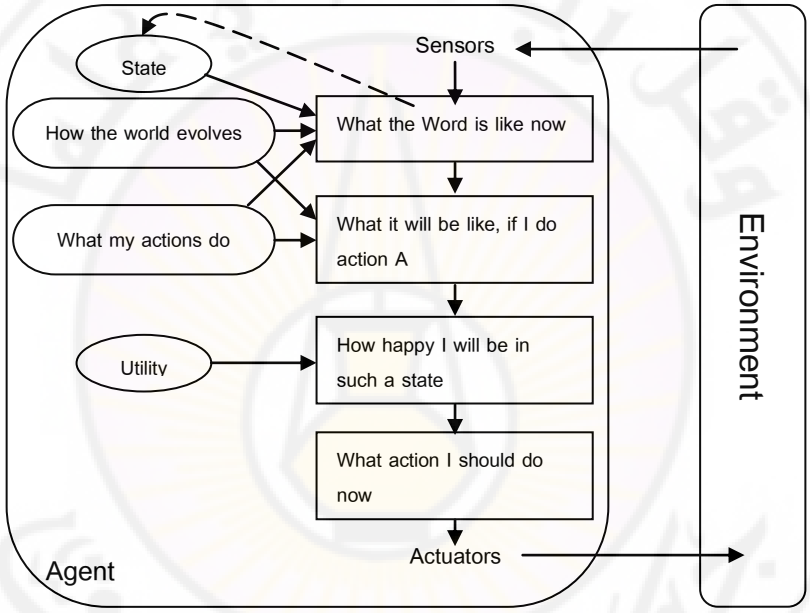
معرفة الحالة الراهنة للوسط ليس كافياً لتقرير ما يجب فعله، فتقرير ما يجب فعله يعتمد على الهدف، كما في الشكل(2-5). بالإضافة إلى معلوماته عن البيئة الراهنة فإنه لديه معلومات عن الهدف المراد الوصول إليه، ولديه مجموعة أهداف جزئية كلما حصل عليها توصل للهدف بشكل أفضل . ما يميز هذا النوع هو الاعتبارات المستقبلية في عملية اتخاذ القرار، أي الهدف بدل قاعدة المعرفة، كالسيارة الآلية. وبالتالي فهو أقل فعالية ولكنة أكثر مرونة.



الشكل(2-5) وكيل ذو هدف معين

2-7-4- وكيل قائم على التفضيل (Utility-Based Agents):

يستخدم عند وجود أكثر من طريقة للوصول إلى الهدف ، كما في الشكل(2-6)، ويفاضل بين الطرائق المختلفة وإيجاد الطريق الأكثر فائدة (الأقصر، الأمثل). في هذا الوكيل يكون الاعتماد على الفائدة أكثر من الهدف.



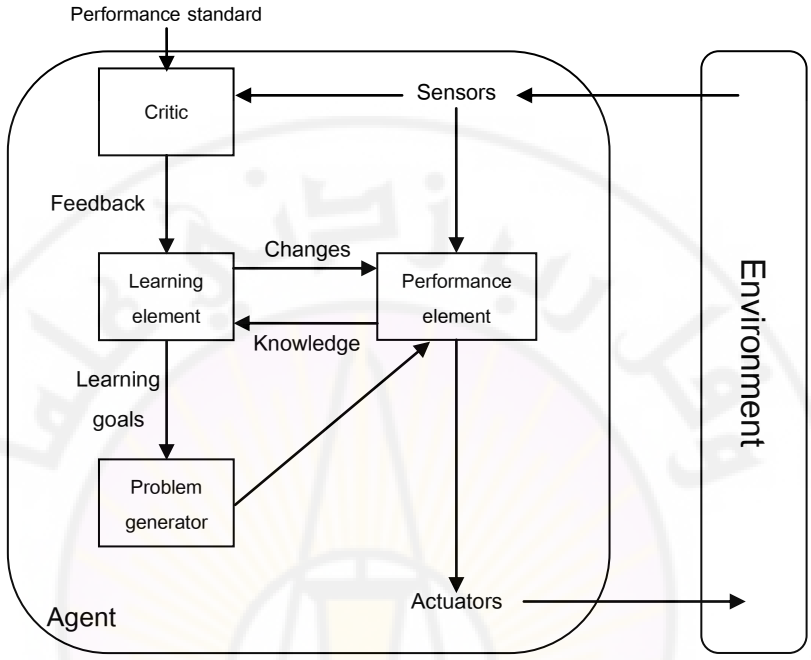
الشكل(2-6) وكيل قائم على التفضيل

2-7-5- وكيل قائم على قابلية التعلم (Learning Agents):

تبنى الأنواع السابقة على أساس اختيار الوكيل للإجراء المجهز سلفاً، فهو مجهز للاختيار فحسب. أما الوكيل القائم على التعلم فهو مجهز بطريقة يستطيع أن يتعلم من خلالها ، كما في الشكل(2-7)، ويعد أكثر الوكلاء شبهاً بالإنسان ، ويستخدم في الأوساط التي لا نعرف عنها الكثير. الفائدة الرئيسة من هذا الوكيل

هو القدرات الفائقة على التأقلم مع وسط غير معروف سلفاً ، فهو يتعلم من الوسط الذي يتواجد فيه. الوكيل المبني على قابلية التعلم.

للقيام بعملية التعلم يضاف ثلاث كتل في مخطط الوكيل وهي، (Critic, Learning Element, Problem Generation). في مثال سيارة الأجرة ذاتية القيادة، عندما يشاهد الوكيل فراغ على يمين السيارة التي أمامه ويتجاوزها، فإن (Critic) يخبر أن الإجراء الذي تم القيام به غير صحيح ، ويتم تعديل قاعدة المعرفة بعدم التجاوز من اليمين مرة أخرى من خلال (Learning Element). أما مفهوم (Problem Generation)، فهي طريقة التعلم الذاتي من خلال تجربة، كالحفاظ على السرعة المرتفعة عند المطبات واهتزاز السيارة نتيجة لذلك، فهنا تكتسب معرفة جديدة لتخفيف السرعة عند المطبات اللاحقة. وهكذا يتم تجميع معارف متنوعة وبسيطة للحصول على العلم.



الشكل(2-7) وكيل قائم على قابلية التعلم

يهمنا في النهاية عملية التصميم، حيث يجب دراسة الوكيل الأفضل للقيام بعملية التصميم. فمثال الروبوت الذي يطوف حوالي الغرفة بجانب الحائط ، يكون الوكيل المناسب له هو وكيل يعتمد على نموذج وكيل، أي له ذاكرة.

الفصل الثالث

3- تمثيل المعرفة (Knowledge Representation):

3-1- مقدمة:

لكي يتمكن الحاسوب من حل مسألة ما، لا بد له من فهمها أولاً، وهذا ما نسعى إليه. حيث يجب تلقين الحاسوب المعطيات والشروط والفرضيات الخاصة بالمسألة، وبالطبع، الهدف الأساسي منها، أي الحل الذي نسعى له. وبالنتيجة، نقل المعارف التي لدينا، بصيغة مناسبة يفهمها الحاسوب.

سوف نتعرف على بناء المعرفة داخل نظام الحاسوب، وفق أكثر الطرائق استخداماً في تمثيل المعرفة كالمنطق الرياضي، والشبكة اللفظية، والوراثة، وطرائق تمثيل المعارف غير الدقيقة. حيث نتناول طرائق تمثيل المعرفة المرتبطة بطبيعة المشكلة مع الأخذ بالاعتبار أنه ليست هناك طريقة محددة لتمثيل المعرفة المتعلقة بمشكلة خاصة. بعد ذلك ننتقل إلى تمثيل المعرفة بالمنطق الرياضي، وهنا يتم بناء قواعد المعرفة عبر تخزين مجموعة جمل أو عبارات لغوية مصاغة صياغة سليمة، ثم يتم معالجة الجمل بطريقة تجعلها قابلة لأن تمثل داخل الحاسوب، والجمل هنا إما أن تقدم أخباراً كحقائق أو قواعد. وهذه الحقائق أو القواعد إما أن تكون صحيحة أو خاطئة. بعد ذلك نتناول تمثيل المعرفة بالشبكات اللفظية التي تقدم تمثيلاً جيداً لكل من أجزاء الكائنات، والتصنيفات الموجودة في هذه الكائنات، والعلاقات بين الكائنات.

يقوم علم الذكاء الصنعي على محورين أساسيين هما تمثيل المعرفة ، والبحث. وتتكون معظم نظم الذكاء الصنعي من جزأين هما:

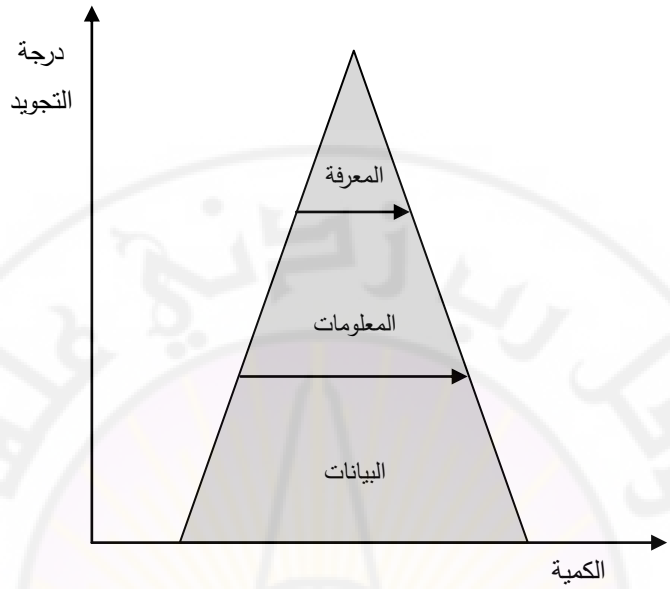
- قاعدة المعرفة (Knowledge Base): هي مجموعة المعارف المتعلقة بمسألة معينة.

- آلة الاستدلال (Inference Engine): هي عملية استخلاص حقائق جديدة من حقائق معطاة.

يجب التمييز بين البيانات (Data)، والمعلومات (Information)، والمعرفة (Knowledgebase)، حيث أن:

- البيانات: هي المادة الخام من المعلومات ، كالحروف والأرقام والرموز والمشاهدات والحقائق والملاحظات التي يتم إدخالها للحاسوب.
- المعلومات: هي ناتج تحليل البيانات ، وذلك بغرض استخلاص العلاقات والمقارنات والمؤشرات ومعاملات الارتباط، والتي على ضوئها تتخذ القرارات. وبذلك يمكن القول إن المعلومات تبدأ من حيث تنتهي البيانات.
- المعرفة: هي محصلة الامتزاج الخفي لعناصر ثلاثة هي، المعلومات والخبرة والحكمة البشرية . بمعنى أبسط، تبدأ المعرفة حيث تنتهي المعلومات.

يبين الشكل(3-1)، تمثيلاً مبسطاً للعلاقة بين البيانات ، والمعلومات، والمعرفة.



الشكل(3-1) العلاقة بين البيانات والمعلومات والمعرفة

3-2- تمثيل المعرفة:

مما سبق نستطيع أن نقول، إنه ليست هناك طرائق محددة لعلاج مشاكل الذكاء الصنعي، حيث تعالج مشاكل الذكاء الصنعي بصورة مفتوحة دون إتباع أساليب مقيدة، وهذا ما يراه البعض أحد مميزات الذكاء الصنعي.

لاقت بعض طرائق تمثيل المعرفة اهتماماً خاصاً ، ويمكن أن تكون أكثر الطرائق استخداماً في تمثيل المعرفة، كالمنطق الرياضي (Mathematical Logic)، والشبكة الدلالية أو اللفظية (Semantic Network)، والأطر الجدولية ، والوراثة (Inheritance)، بالإضافة إلى طرائق تمثيل المعارف غير الدقيقة.

تصنف طرائق تمثيل المعرفة إلى طرائق مرتبطة بمكونات بيئة المعرفة، وطرائق مرتبطة بطبيعة المشكلة، وتعد الأخيرة أكثر الطرائق استخداماً في تمثيل

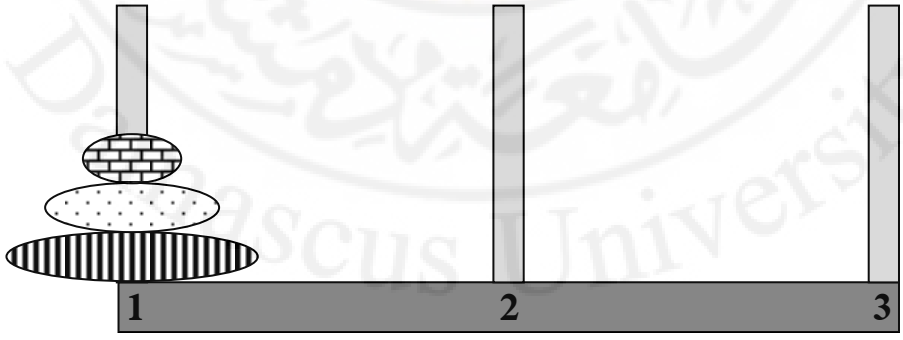
المعرفة ، ويتطلب تمثيلها خطوات ومنهجية خاصة بهذه المشكلة، ما يستوجب طرائق خاصة لتمثيل كل نوع من المشاكل، حيث لا يكون هناك طريقة محددة لتمثيل المعرفة المتعلقة بمشكلة خاصة. فمثلاً عند معالجة لعبة تتطلب ذكاء داخل الحاسوب، تختلف طريقة تطبيق المعارف الخاصة بهذه اللعبة عن بقية الألعاب الأخرى، ولكن تتفق كلها في الهدف الذي تنشده كل طريقة، وهو حصر جميع الفرص الممكنة للحركات المتوقعة في أثناء اللعب وتمثيل هذه الفرص داخل الحاسوب. أيضاً، نجد جوانباً خاصة بتمثيل المعارف المتعلقة بمعالجة اللغات الطبيعية تختلف عن تلك التي تعالج نوعاً من أنواع النظم الخبيرة.

- **لعبة أبراج هانوي:**

الهدف من اللعبة كما يبين الشكل(3-2)، هو نقل الأقراص بالكامل من العمود الأول إلى العمود الثالث مع إتباع القواعد الآتية:

- يسمح بنقل قرص واحد فقط بين أي عمودين في كل خطوة.
- لا يسمح بوضع قرص فوق قرص آخر يصغره حجماً.

يمكن حصر جميع الفرص الممكنة لإجراء هذه النقلات ، لنجد أنه يوجد فرصة واحدة ممكنة فقط تؤدي للهدف.



الشكل(3-2) لعبة أبراج هانوي

مما سبق نستطيع أن نقول، إنه ليست هناك طرائق محددة لعلاج مشاكل الذكاء الصنعي، حيث تعالج مشاكل الذكاء الصنعي بصورة مفتوحة دون إتباع أساليب مقيدة، وهذا ما يراه البعض أحد مميزات الذكاء الصنعي.

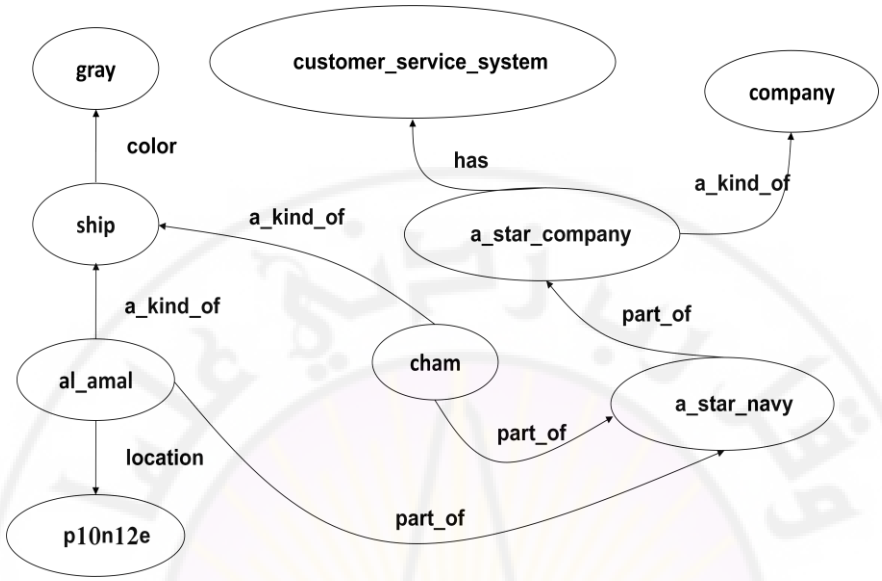
3-3- الشبكات الدلالية:

تعد الشبكات الدلالية من الطرائق الرسومية للتمثيل وهي أقدم طرائق تمثيل المعرفة وأوضحها حيث تتكون من مجموعة من العقد والروابط التي تظهر العلاقات الهرمية بين الأشياء.

تتكون الشبكات الدلالية مما يأتي:

1. العقد: يرمز لها على شكل دوائر ، وهي تمثل أشياء ومعلومات وصفية عن هذه الأشياء.
2. الأقواس: تتصل العقد ببعضها بروابط أو أقواس ، وهذه الأقواس توضح العلاقات بين الأشياء.

يبين الشكل(3-3) شبكة دلالية لشركة نقل بحري.



الشكل(3-3) شبكة دلالية لشركة نقل بحري

3-4- تمثيل المعرفة بالمنطق الرياضي (Mathematical Logical):

تتمثل المعرفة لدى البشر من خلال تقويم للغات الطبيعية للبشر، حيث كل عبارة من عبارات لغات البشر تعطي نتائج عن حقائق أو قواعد يخزنها العقل البشري مكوناً منها معرفته . ولتطبيق المعرفة داخل الحاسوب، لا بد من معالجة اللغات الطبيعية.

إن بناء قاعدة المعرفة يتم عن طريق تخزين مجموعة من جمل أو عبارات لغوية مصاغة صياغة سليمة ، ثم معالجة الجمل بطريقة تجعلها قابلة لأن تمثل داخل الحاس وب. تكون الجمل أو العبارات في اللغات الطبيعية وفق منطق متعارف عليه في تركيب الجمل كالإعراب (Syntax)، والمعنى (Semantics).

تعد الجملة الصحيحة في تركيبها ومعانيها ، مصاغة صياغة جيدة ويرمز لها بالرمز ((Well-Formed Formula (WFF).

واحدة من الطرائق المستخدمة لتحويل هذه الجمل إلى صيغ يتم عبرها بناء المعرفة داخل الحاسوب هو استخدام المنطق الرياضي، لما يوفره المنطق الرياضي من إمكانيات تعبر عن منطق هذه الجمل.

والجمل (WFF) وفق المنطق الافتراضي (Propositional logic) إما أن تكون جملاً بسيطة أو جملاً مركبة ، كما يتضح ذلك من خلال رموز باكيوس المعروفة بالرمز ((Backus-Naur Form (BNF).

إذا أخذنا المعاني التي نستخلصها من هذه الجمل نجدها إما أن تقدم أخباراً كحقائق أو قواعد، وهذه الحقائق أو القواعد ، إما أن تكون صحيحة أو خاطئة (True or False).

ما هي الحقائق (Facts)؟

هي الجمل التي تصف شيئاً ما، أو تخبر عن شيء ما ، سواء أكانت هذه الجملة اسمية أم فعلية أم شبه جملة ، وقد تكون صحيحة (True) أو خاطئة (False).

فعلى سبيل المثال، الجمل الآتية هي حقائق صحيحة:

Artificial intelligence is a computer course.

Cat is an animal.

وقد تكون الحقيقة مركبة مثل:

John's mother is married to John's father true

أما الجملة الآتية فهي حقيقة خاطئة:

Cat is a human

ما هي القواعد (Rules)؟

هي الجمل التي يمكن تعميمها أو تطبيقها على مجموعة من الأشياء ، ويلزم تطبيقها توفر شرط أو مجموعة شروط.

وعادة تكون القاعدة في صورة جملة (If) مثل:

If animal gives milk it is a mammal

3-5- فئات الكلمات في المنطق الرياضي:

إذا لاحظنا العبارات السابقة، نجد أن الكلمات التي تكون هذه الجمل إما أن تكون أسماء أو أفعالاً أو صفات أو حروف ربط، وهذه التصنيفات للكلمات من منظور رياضي يمكن تصنيفها في (6) فئات هي:

1. المسند (predicate): هي الكلمات التي تصف العلاقات بين المحددات أو الوسطاء (arguments)، كتحديد خاصية محدد أو ارتباط عدة محددات مع بعضهم، مثل:

   father(Asaad, Ousama), city(Damascus)

2. الثوابت (constant): يرمز لها غالباً بأحرف كبيرة، مثل:

   Syria, M, 5

3. الدوال (functions): تشير للعلاقة الوظيفية، مثل:

   father of(x), age(x), sin(30)

4. المتغيرات (variable): يرمز لها غالباً بأحرف صغيرة، مثل:

x, a, var1

5. أدوات الوصل أو الروابط (connective): مثل:

→, ∨, ∧, ¬

6. محددات الكميات (quantifier): للتحويل من المنطق الافتراضي إلى المنطق الرياضي، يوجد محددان هما:

- محدد الشمولية (∀ Universal quantification).
- محدد الوجودية (∃ Existential quantification).

إن الشكل العام لتركيب الجمل وفق المنطق الرياضي هو:

Predicate name(Argument 1, …. , Argument n)

حيث العناصر التي بين القوسين (Argument) تمثل بقية الكلمات خلاف المسند.

1. Computer_Course(Artificial_Intelligence)
2. Animal(CAT)
3. Married(mother, (John), father, (John))
4. Come(Easy), Go(Easy)
5. Communication(Evil), Corrupt(Good_Manners)
6. Every(Why), Has(Answer)
7. Animal (give (Milk)), Mammal (Animal)

من الصيغ أعلاه نلاحظ الآتي:

إن تحويل الجمل للصيغة أعلاه يأخذ إحدى صورتين، الأولى صورة غير مركبة وتتمثل في الجمل (1, 2)، والثانية صورة مركبة وتتمثل في الجمل (3, 7)، وذلك لأن أصل الجمل التي أتت منها هذه الصيغ إما جمل غير قابلة للتجزئة (Atomic WFF) والتي كان نتاجها العبارات في الجمل (1, 2)، أو جمل مركبة (Compound WFF) والتي أتت منها بقية العبارات.

إن هناك كلمات كتبت بـأحرف كبيرة أو يمكن فقط كتابة حرفها الأول بأحرف كبيرة، وأخرى كتبت بـأحرف صغيرة، وذلك لأن هناك قاعدة تم التعارف عليها في صياغة تلك الجمل وهي:

- المسند (Predicate): يكتب أو يبدأ بأحرف كبيرة.
- العناصر (Argument): إذا لم تكن أسماء تكتب بأحرف صغيرة، أما إذا كانت أسماء فتكتب بأحرف كبيرة.

وحتى يتم تمثيل هذه الجمل وفق المنطق الرياضي ، فلا بد من تحويلها إلى رموز مع استخدام الرموز الرياضية المعروفة:

الرمز (Symbol)	المعنى (Meaning)
∀	مهما يكن أو لكل (For All)
∃	يوجد على الأقل (Exist)
¬	النفي بلا (Not)
∧	و (AND)
∨	أو (OR)
→	إذن (Then)
gt	أكبر من (Greater than >)
lt	أقل من (Less than <)

ge	أكبر من أو يساوي (Greater than or equal >=)
le	أصغر من أو يساوي (Less than or equal <=)
=	يساوي (Equal =)

والآن لنرى كيف يمكن وضع الجمل في صيغ المنطق الرياضي. إذا أخذنا مثلاً الجملة:

If animal gives milk it is a mammal

ANIMAL(give (MILK)), MAMMAL (ANIMAL)

تكون الجملة على النحو الآتي:

$$\forall x\{ANIMAL(x) \wedge give(x,y) \Rightarrow MAMMAL(X)\}$$

نحول الجملة إلى علاقة منطقية ينتج لنا:

$$(X \wedge Y) \Rightarrow (Z \wedge X)$$

وإذا كونا جدول المنطق لهذه الجملة يكون على النحو الآتي:

X	Y	Z	$X \wedge Y$	$Z \wedge X$	$X \wedge Y \Rightarrow Z \wedge X$
T	T	T	T	T	T
T	T	F	T	F	F
T	F	T	F	T	T
F	T	T	F	F	T
F	F	T	F	F	T
T	F	F	F	F	T
F	F	F	F	F	T

لنأخذ مثالاً أكثر تعقيداً.

الجملة المركبة الآتية:

Every city has a dogcatcher who has been bitten by every dog in that city.

إذا رمزنا للمدن (x) ولصيادي الكلاب (y) وللكلاب (z) تكون الجملة على النحو الآتي:

$$(\forall x)\{[CITY(x)\rightarrow(\exists y)(DOGCATCHER(y)\wedge LIVES_IN(x,y))]\wedge(\forall z)[DOG(z)\wedge LIVES_IN(z,x)\rightarrow BIT(z,y)]\}$$

نلاحظ أن تحويل الجمل الطبيعية، وإن كان يتبع قاعدة معينة، إلا أنه يعتمد على الخبرة، وقد يختلف اثنين في صياغة الجمل منطقياً. ولكن عند تصميم جدول المنطق تتفق الحلول جميعها في النتيجة.

كما نلاحظ أنه يمكننا تحويل أي جملة إلى صياغة وفق المنطق الرياضي ومفهوم المسند، و يمكننا أن نولد مجموعة حقائق من حقيقة معطاة، أو من خلال الدمج بين مجموعة جمل.

نبين فيما يأتي بعض الجمل وصياغتها باستخدام المنطق الرياضي (Predicate Logic).

Cat is an animal

Animal (CAT)

Dog is an animal

Animal (DOG)

Dogcatcher is a human

Human (DOGCATCHER)

Person is human

Human (PERSON)

Animal has 4 legs

Leg(animal)=4

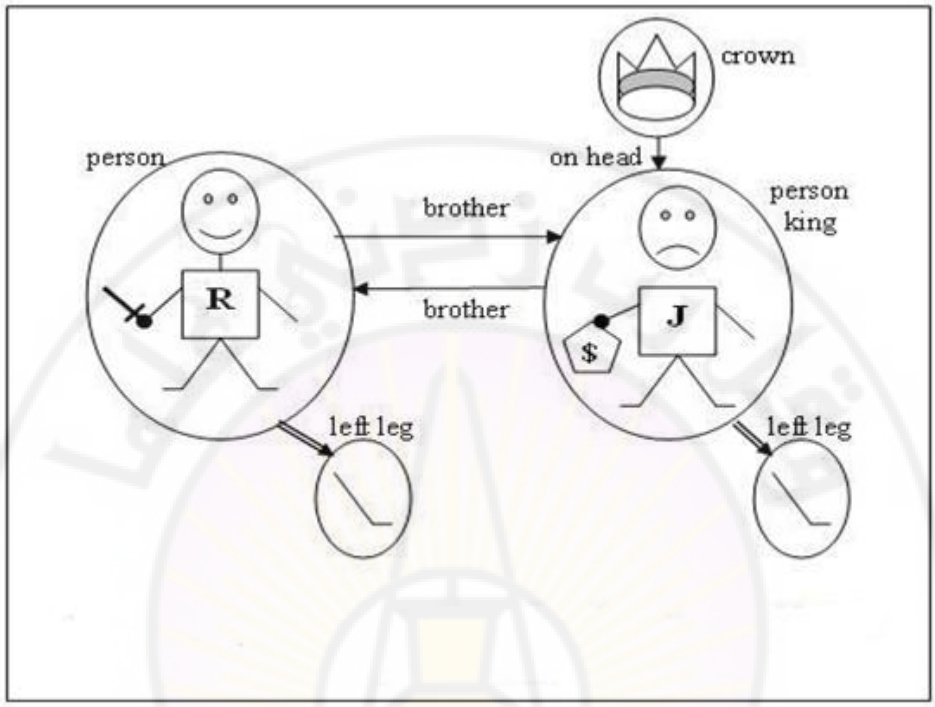
هذا يؤدي إلى أن:

Leg(DOG)=4, Leg(CAT)=4

مثال:

لنأخذ الآن مثالاً شاملاً نتدرج به حتى نصل إلى الصورة النهائية.

يتعلق المثال بملك بريطانيا ريتشارد قلب الأسد الذي حكم لفترة (10) أعوام من عام (1189) إلى عام (1199). وشقيقه الأصغر جون الشرير الذي حكم لفترة (16) عاماً، من عام (1199) إلى عام (1215). هذه القصة يمثلها النموذج الموضح في الشكل(3-4):



الشكل(3-4) نموذج لقصة ريتشارد قلب الأسد

نموذج يحتوي على خمسة كائنات (شخصين، وتاج، ورجلين يسار). يقدم النموذج الموضح تصويراً لقصة ريتشارد قلب الأسد، وشقيقه الأصغر الملك جون.

Richard the Lion heard

King John

ومن الشكل يوجد بالإضافة للكائنين اللذين يمثلان الأشقاء ، يوجد كائن يمثل التاج، والعلاقة هنا هي على رأس الملك (on head)، أي أن التاج على رأس الملك الذي هو جون ، لأن ريتشارد قد مات في ذلك الوقت . أما الكائنين الآخرين فهما (Leftleg) لتوضيح علاقة تمثل دالة (function).

من هذه المفاهيم نستطيع أن نستنتج عدة قواعد ، ونتتبعها لنوضح كيف يمكن أن نستخلص قواعد معرفية وفق طريقة المنطق الرياضي.

1. الجمل المنطقية البسيطة (Atomic Sentences):

يمكن تركيب العديد من الجمل البسيطة مثل:

Brother(Richard, John)

Married(Father(Richard), Mother(John))

2. الجمل المركبة (Complex Sentences):

إذا استخدمنا أي من الروابط التي تربط بين جملتين ، تتكون معنا جمل مركبة مثل:

¬Brother(LeftLeg(Richard), John)

Brother(Richard, John) ∧ Brother(John, Richard)

King (Richard) ∨ King (John)

¬King(Richard) → King(John)

3. استخدام محددات الكميات (Quantifiers):

- محدد الشمولية (الإجمالي):

في هذه الحالة يمكننا أن نأخذ المسألة بصورة شاملة كأن نقول، كل ملك إنسان:

All kings are person

∀ x King(x) → Person(x)

نلاحظ أننا استخدمنا في هذه الحالة المتغيرات (x) والعبارة تعني ، لكل (x) إذا كان (x) هو ملك إذن (x) هو شخص.

(For all x, if x is a king, then x is a person)

ملاحظة :

السهم (→) يؤدي هو أداة الوصل الرئيسية مع المحدد الكلي (∀).

لا تستخدم (∧) كأداة وصل رئيسية مع (∀).

- محدد الوجودية (الجزئي):

في هذه الحالة يمكننا أن نعبر عن شيء خاص كأن نقول، للملك جون تاج على رأسه:

King John has a crown on his head

∃ x Crown(x) ∧ On Head (x, John)

نلاحظ إننا استخدمنا أيضاً المتغيرات (x) والعبارة تعني أن هناك (x) حيث (x) هذه تاج، وهو على رأس جون.

There exist an x such that x is crown on Jon head.

ملاحظة:

الجمع المنطقي (∧) هو أداة الوصل الرئيسية مع المحدد الجزئي (∃)

لا تستخدم السهم (→) يؤدي كأداة وصل رئيسية من (∃).

الجمع بين محددات الكميات (Nested Quantifiers):

كمثال إذا رغبنا أن نقول كل الأشقاء أخوان:

brothers are sibling

∀ x ∀ y Brother(x, y) ∧ Sibling (x, y)

وهذه الجملة يمكن أن تكتب:

∀ x, y Brother(x, y) ∧ Sibling (x, y)

كما يمكن أن نجمع بين نوعي المحددات ، كمثال، لكل شخص ، شخص يحبه:

Everybody loves somebody

$\forall x \exists y$ Loves(x, y)

وتعني هذه الجملة: لكل شخص (x)، هنالك شخص ما (y)، يحبه هذا الشخص.

4. المساواة (Equality):

كأن نقول مثلاً:

Father(John) = Henry

كما يمكن أن تستخدم المساواة لتثبيت حقائق مثل:

$\exists$ x, y Brother(x, Richard) $\wedge$ Brother(y, Richard) $\wedge$ not (x = y) not (y = x) .

وهذا يعني أن لريتشارد شقيقين، وذلك يستنتج من العلاقة.

وعموماً فإن المحددات تستخدم للتحويل من المنطق الافتراضي (الاقتراحي) (Propositions Logic) إلى المنطق الاسنادي (Predicate Logic). ومن مزاياها أنها معبرة جداً، ولديها بناء جملة ودلالة لا لبس فيه. لكن من مساوئ المحددات أنه لا يوجد إجراءات فعالة بشكل عام لمعالجة المعرفة.

من هذه المفاهيم يمكن تمثيل أي مجموعة معارف وتحويلها إلى صيغ يسهل تطبيقها وتمثيلها داخل أنظمة الحاسوب. حيث تبنى قواعد المعرفة بتحويل العبارات المنطقية إلى جمل برمجية بإحدى لغات برمجة الذكاء الصنعي المعروفة مثل (Prolog) و(Lisp)، أو اللغات المتعددة الإمكانيات مثل (Java) و(C++).

وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الرابع

4- تقنيات البحث (Search Technique):

4-1- مقدمة:

في تطبيقات الذكاء الصنعي عادة ما يتم الحصول على الحل عبر تتبع سلسلة من الإجراءات والنتائج. هذه السلسلة من الخطوات غير المعروفة مسبقاً ، للعثور على الحل الذي يطابق مجموعة من المعايير من بين مجموعة من حلول محتملة، هي ما تعرف بخوارزمية البحث. من الإجراءات والنتائج تعرف بعملية البحث. حيث تتم عمليات الاستنتاج والاستفسارات عن المعلومات الموجودة داخل قواعد المعرفة عن طريق البحث عن القيم التي تحقق النتائج المطلوبة، وعادة ما يتولد من الاستفسار عدة نتائج أو تكون هناك أكثر من مرحلة يمر بها الاستنتاج حتى نصل إلى الهدف النهائي. هذه الخطوات والإجراءات كما ذكرنا أعلاه تعرف بالبحث، وهو يأخذ أشكالاً وتقنيات مختلفة. يقوم مبدأ عمل خوارزميات البحث في أنها تأخذ المشكلة كمدخلات ، ثم تقدم الحل في صورة سلسلة من العمليات والأحداث المستمرة ، التي تنتهي عند الوصول إلى الهدف والحصول على الحل النهائي.

وجدت خوارزميات البحث في الذكاء الصنعي، لحل المشكلات ذات الصفات الآتية:

- لا يوجد للحل طريقة تحليلية واضحة، أو أن الحل مستحيل بالطرائق العادية.
- يحتاج الحل إلى عمليات حسابية كثيرة ومتنوعة لإيجاده كمسائل الألعاب والتشفير.

- يحتاج الحل إلى حدس عالي كلعبة الشطرنج.

من هذا المنطلق يمكن القول أن الذكاء الصنعي يقوم بحل المشاكل عن طريق البحث عن الحلول في فضاء الحالة (State Space)، وهذا البحث يتم بعدة طرائق وتقنيات. فما هو فضاء الحالة؟ وما هي هذه التقنيات؟ وكيف تعمل كل تقنية من هذه التقنيات؟.

نهتم في القسم الأول من هذا الفصل بفضاء الحالة، وهنا نبحث عن حل المشكلة المحددة عبر فضاء الحالة . كما نتناول شجرة البحث لإيجاد الطريق من (X) إلى (Y)، في هذه الحالة يكون الحل هو البحث عبر الطرائق المختلفة انطلاقاً من (X) حتى نصل إلى الهدف (Y). كما يتم التطرق إلى مقارنة بين فضاء الحالة وشجرة البحث.

نتناول في القسم الثاني خوارزمية البحث ، حيث نضع إجابة للتساؤلين الآتيين:

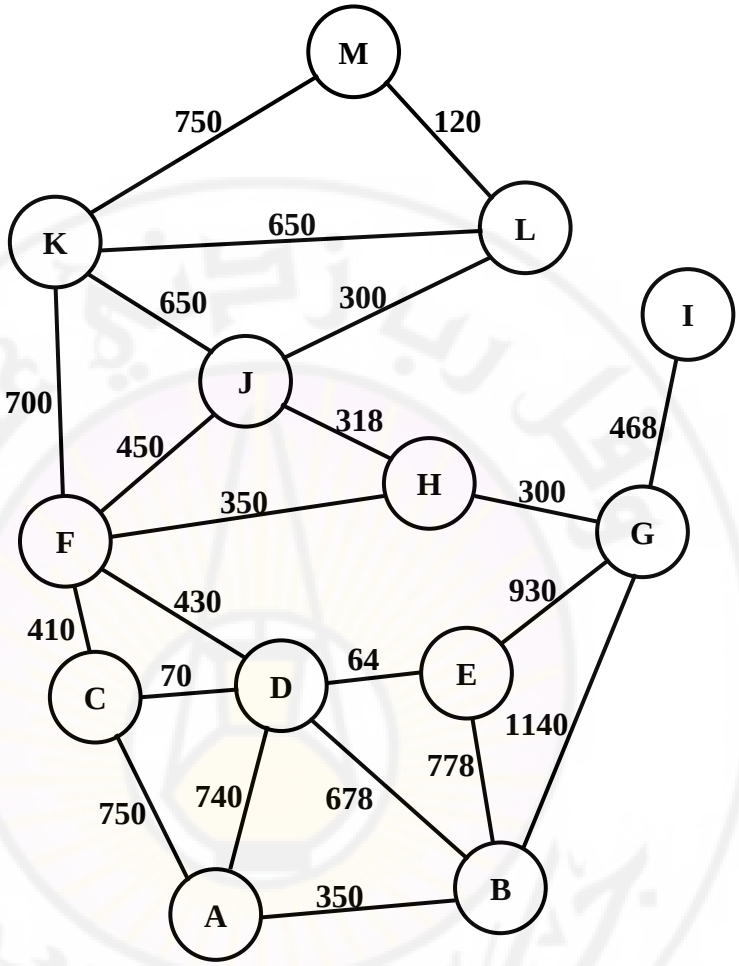
- كيف نعرف أننا وصلنا إلى الهدف؟
- كيف نختار النقطة التالية؟

نتحدث في القسم الثالث عن خمس من استراتيجيات البحث الأعمى أو غير المعلوم ، والذي يعتمد على استخدام أدوات خاصة تختلف باختلاف استراتيجيات البحث.

في القسم الرابع نتطرق للبحث التجريبي أو المعلوم، وهنا سنتناول طريقة مهمة تعرف بالبحث عن الأفضل أولاً، والتي تقوم على اختيار أفضل النقاط لتكون النقطة التالية، كما سنتطرق أيضاً إلى طريقة الأفضل أولاً الطماع، وطريقة (A*).

4-2- فضاء الحالة (State Space):

إذا كان لدينا مشكلة محددة ونبحث عن حل لهذه المشكلة ، فإن الحل يتم بالبحث عبر فضا ء الحالة. فمثلاً ، إذا كنا نرغب أن نجد الطريق من مدينة إلى أخرى داخل دولة معينة ، فإن فضاء الحالة يكون كل المدن التي توجد في هذه الدولة، أما الهدف فيكون هو الوصول إلى المدينة (F) انطلاقاً من المدينة (A). وإذا تمعنا في مثل هذا المثال نجد أن فضاء الحالة يمكن أن نعتبره كل النقاط التي تمثل المدن في شكل (Graph)، يحتوي على نقاط (Nodes) هي المدن ، وتربط بينها خطوط (Edges) هي الطرق التي تربط بين هذه المدن. يبين الشكل(4-1) بعض المدن في الدولة (X)، والطرق التي تربط بين هذه المدن . فضاء الحالة في هذا المثال هو عبارة عن (13) نقطة هي مجموعة المدن الموجودة في الشكل. وقد تم وضع المسافة بين المدن لاستخدامها في حساب الكلفة.



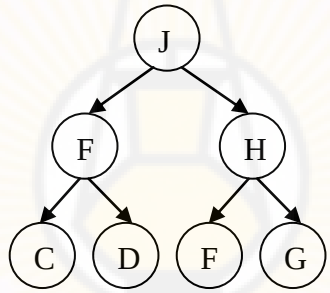
الشكل(4-1) نموذج لبعض المدن والطرق التي تربط بينها في الدولة (X)

4-3- شجرة البحث (Search Tree):

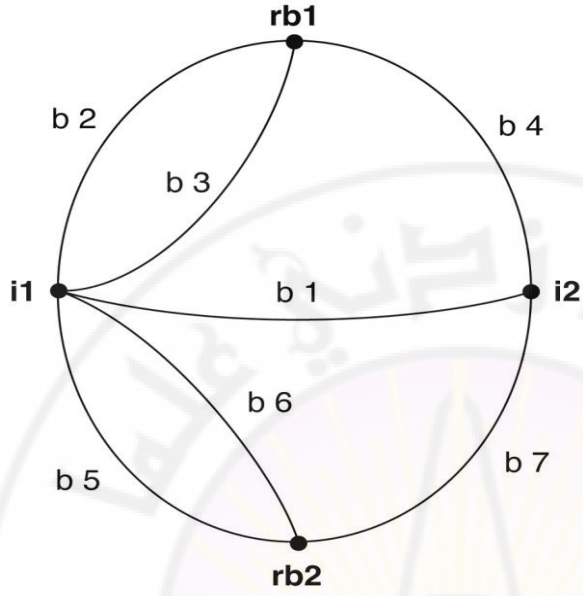
إذا كان المطلوب إيجاد الطريق من (J) إلى (A)، ففي هذه الحالة يكون الحل هو البحث عبر الطرق المختلفة انطلاقاً من (J) حتى نصل إلى الهدف (A). ومن هنا نلاحظ أن البحث يأخذ شكل شجرة بحث جذرها (J) وتتفرع منها

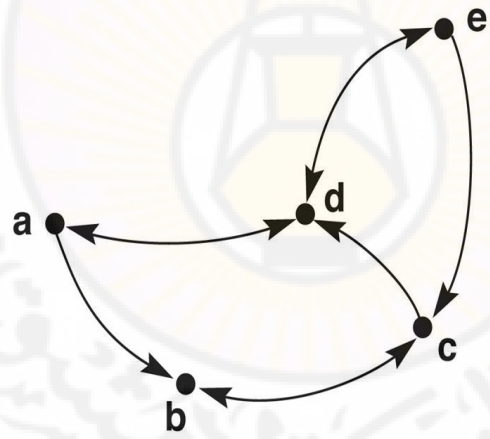
عدة فروع كما هو موضح في الشكل(4−2). إن إتباع أي واحدة من هذه الفروع يولد ممراً (Path) من ممرات الشجرة أو يولد شجيرات جديدة.

نلاحظ من الشكل(4−2) أننا إذا رغبنا في الانتقال من (J) إلى (A) فإنه لدينا عدة خيارات في كل مرحلة، وأننا كل ما وصلنا مرحلة من المراحل تحذف خيارات وتتولد خيارات أخرى، وكل مرحلة تمثل شجرة بحث جديدة بجذر جديد. وهذا يعني أن هناك عدداً كبيراً من الممرات (Paths) في الدولة للانتقال عبر الشكل الكلي الممثل في الشكل(4−1) وقد يصل هذا العدد إلى ما لا نهائي من الممرات أو الشجيرات المتجددة، وذلك لأن البحث في الأصل ليس بحثاً شجرياً وإنما هو بحث في شكل (Searchgraph).



الشكل(4−2) شجرة البحث، جذر الشجرة مدينة (J)





Nodes = **{a,b,c,d,e}**

Arcs = **{(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}**

الشكل(4-3) مخطط الحالات

يتكون المخطط من مجموعة من العقد (Nodes) التي تمثل حالات متقطعة (Discrete State)، ومجموعة من الأقواس (Arcs) أو الوصلات (Links) التي تصل بين كل زوجين من العقد . وكمثال على ذلك مثل مسألة جسور كونيغسبرغ السبعة (Seven Bridges Königsberg)، حيث تمثل العقد البر والوصلات الجسور.

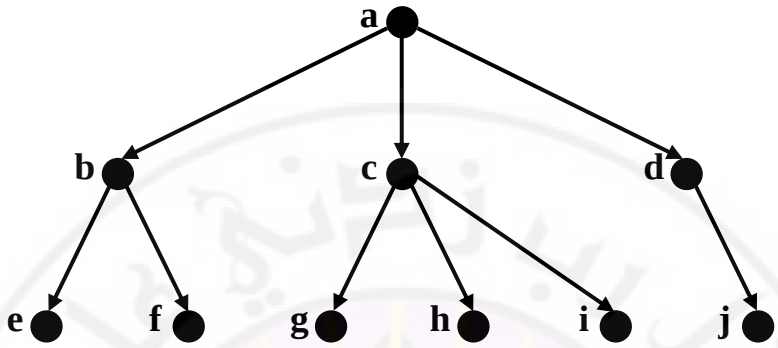
يكون المخطط موجهاً إذا كانت الأقواس مرتبطة باتجاه محدد . ترسم الأقواس في المخطط الموجه كأسهم كما في الشكل(4-3). يصل المسار (Path) خلال المخطط بتتابع من العقد عبر أقواس متتالية. يتم تمثيل المسار بقائمة مرتبة تسُجل العقد فيها بترتيب ظهورها في المسار. في أعلاه يمثل [a,b,c,d] المسار عبر العقد (a) ثم (b) ثم (c) ثم (d) بالترتيب نفسه.

يقال عن المسار الذي يحوي أي عقدة أكثر من مرة أنه يحوي دورة أو حلقة (Cycle or Loop). للمخطط الجذر (Rooted Graph) عقدة وحيد ة تسمى الجذر (Root) حيث يوجد مسار من الجذر إلى كل العقد داخل المخطط ، ويكون الجذر في أعلى المخطط الجذر. فمثلاً، مخططات فضاء الحالة للألعاب تكون عادة مخططات جذر حيث تكون بداية اللعبة هي الجذر.

الشجرة هي عبارة عن مخطط يوجد فيه مسار واحد على الأكثر بين كل عقدتين في المخطط. يكون للأشجار جذور.

يستخدم أسلوب العائلة للعلاقات بين العقد في المخطط الجذر أو الشجرة مثل المخطط في الشكل(4-4).

تسمى عقدتين أنهما متصلتان إذا وجد مسار يحوي الاثنين معاً.



الشكل(4-4) مخطط الجذر

4-4- الفرق بين فضاء الحالة وشجرة البحث:

من الشكلين السابقين نلاحظ أن الانطلاق من مدينة إلى مدينة يتم عبر الانتقال من مرحلة إلى مرحلة، وأن هذا الانتقال يولد عدداً لا نهائياً من الممرات التي تمثل شجيرات البحث. أما المراحل فلا تخرج من النقاط التي تمثل المدن الموجودة والتي عددها (13) مدينة في الشكل(4-1).

هذا يعني أن فضاء الحالة هو مجموعة المراحل الممكن الوصول إليها أو هو النقاط الممكنة في الشكل، وهو محدود بما هو متاح من نقاط أو مراحل يمكن الانتقال إليها. أما شجرة البحث فهي كل طريق أو ممر (Path) يمكن أن نسلكه في الانتقال من مرحلة إلى أخرى. وهو لا يكون بعدد المراحل ، وإنما بعدد ممرات الدولة والتي قد يصل عددها إلى ما لانهاية.

4-5- خوارزمية البحث:

عرفنا سابقاً أن البحث يكون بتحديد نقطة الانطلاق كجذر لشجرة تتفرع منها ممرات متعددة في اتجاه الهدف. نختار من هذه الممرات إحدى النقاط لتكون

نقطة انطلاق جديدة لمرحلة جديدة، وبالتالي تكون هذه النقطة جذراً لشجرة جديدة، وهكذا حتى نصل إلى الهدف. وهذا يقود للتساؤلين الآتيين:

- كيف نعرف أننا وصلنا للهدف؟
- كيف نختار النقطة التالية؟

لمعرفة وصولنا للهدف فإنه علينا ابتداءً من نقطة البداية تكرار اختبار النقطة أو المرحلة الحالية هل هي هدف أم لا، فإذا كانت هدفاً حصلنا على الحل، وإلا نبحث عن النقطة التالية التي ستصبح جذراً جديداً لشجرة جديدة.

ومن هنا فإن مكونات البحث عبارة عن بنية بيانات (Data Structure) تمثل تعريفاً جيداً لمكونات المشكلة على النحو الآتي:

1. المراحل (فضاء الحالة).
2. المرحلة الابتدائية أو نقطة الانطلاق (Initial State).
3. الأحداث الممكنة (Possible Actions): يمكن أيضاً استخدام دالة نجاح (Successor Function) تعمل على المراحل الموجودة، حيث ينتج لدينا سلسلة من الأحداث الممكنة لتحقيق النجاح . ففضاء الحالة الممثل في أي شكل (Graph) يحتوي على مجموعة من المراحل المرتبطة ببعضها عبر خطوط تمثل الأحداث . فالممر (Path) في فضاء الحالة هو سلسلة من الحالات (States) المرتبطة بسلسلة من الأحداث (Action).
4. اختبار الهدف (Goal Test): عند كل مرحلة يتم اختبار هذه المرحلة للتأكد من أنها الهدف أم لا، ويتم هذا الاختبار على كل المراحل التي نمر بها حتى لو كانت المرحلة الأولى.

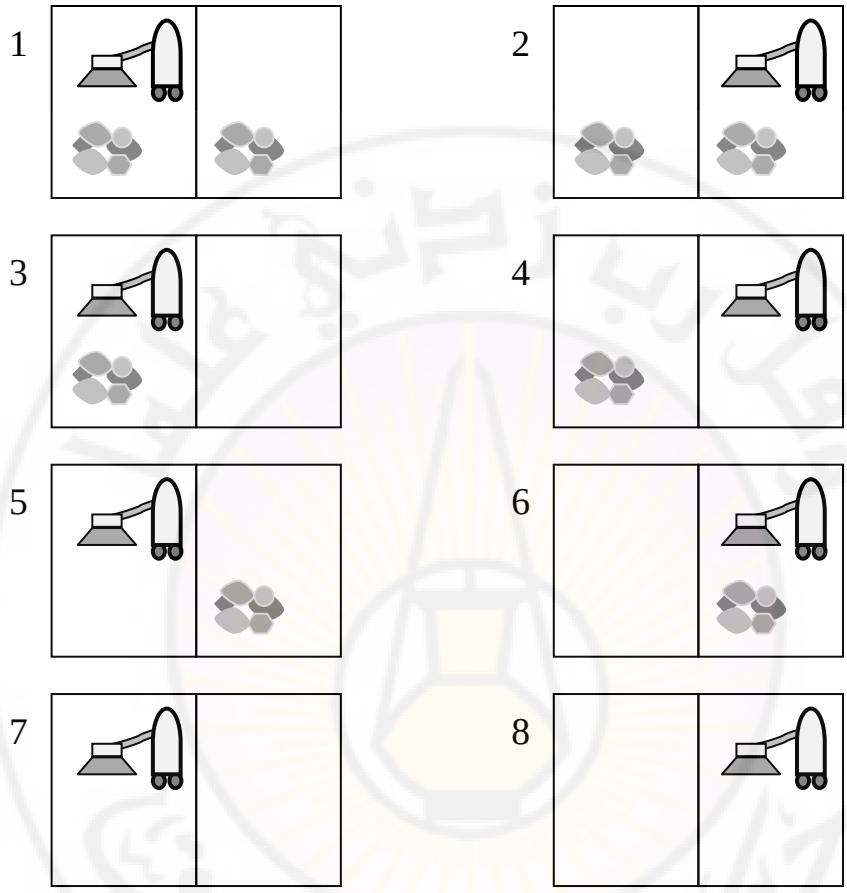
5. تكلفة الممر (Path Cost): وهي الدالة التي تقوم بحساب تكلفة كل ممر من الممرات المتاحة.

حتى نقف على هذه المكونات لنأخذ مثلاً تهيئة مشكلة الانتقال من (J) إلى (A).

1. المراحل: جميع المدن في الشكل(4-1).
2. المرحلة الابتدائية: مدينة (J).
3. الأحداث الممكنة: الانتقال من مدينة لأخرى.
4. اختبار الهدف: في كل مدينة نصل إليها، نقوم باختبارها هل هي (A).
5. تكلفة الممر: نجمع أطوال الطرق من مدينة لأخرى ضمن الممر.

كمثال آخر نأخذ المكنسة الكهربائية في الشكل(4-5):

1. المراحل: ثماني حالات ممكنة.
2. المرحلة الابتدائية: واحدة من الثماني حالات.
3. الأحداث الممكنة: الانتقال لليمين، الانتقال لليسار، شفط الأوساخ.
4. اختبار الهدف: الغرفة نظيفة حالة (7) أو (8).
5. تكلفة الممر: (1).

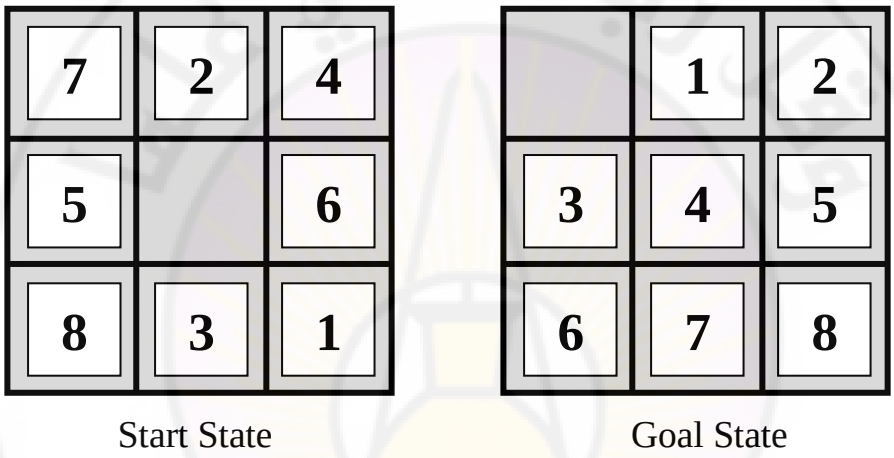


الشكل(4-5) المكنسة الكهربائية

كمثال آخر نأخذ (8-puzzle) في الشكل(4-6):

1. المراحل: مواقع المربعات.
2. المرحلة الابتدائية: مرسومة في الشكل.

3. الأحداث الممكنة : نقل المربع الخالي للأعلى ، أو للأسفل ، أو لليمين، أو لليسار.
4. اختبار الهدف: الحالة الحالية تطابق الحالة النهائية.
5. تكلفة الممر: (1).



الشكل(4-6) تمثيل تنقل المربع الخالي

لننتقل إلى السؤال الثاني المتعلق بكيفية اختيار النقطة أو المرحلة التالية للمرحلة الحالية؟ أي جذر الشجرة الجديدة.

إن اختيار هذه النقطة أو المرحلة يتوقف على نوع خوارزمية أو تقنية البحث المستخدمة، حيث هناك العديد من تقنيات أو خوارزميات البحث المستخدمة في مجال الذكاء الصنعي، ويمكن حصر هذه التقنيات في مجموعتين هما:

1. البحث الأعمى (Blind Search): يسمى أيضاً بالبحث غير المعلوم (Uninformed Search) وفي هذه المجموعة سنتناول خمسة أنواع من استراتيجيات البحث.

2. البحث التجريبي (Heuristic Search): يسمى أيضاً البحث المعلوم (Informed Search).

كما يوجد تصنيفات أخرى للبحث مثل:

1. البحث من البيانات إلى الهدف (Forward).
2. البحث من الهدف إلى البيانات (Backward).

4-6- البحث الأعمى (Blind Search):

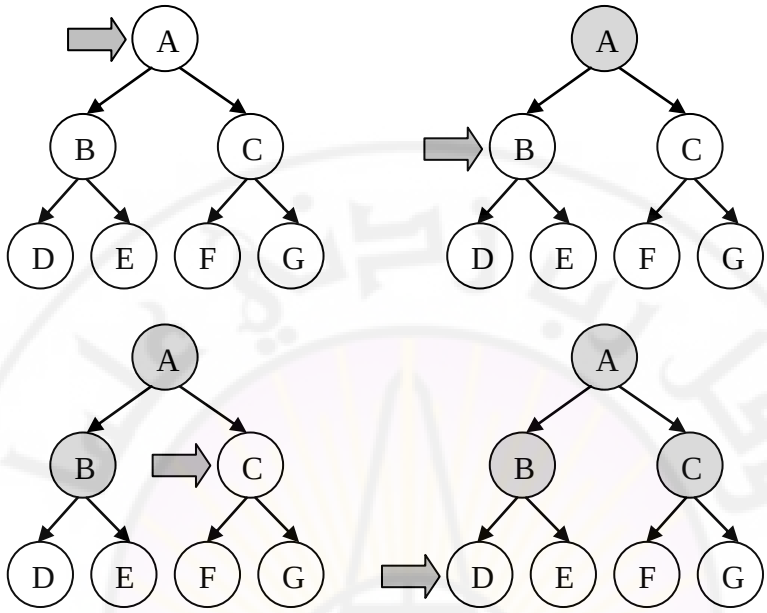
يقوم البحث الأعمى أو البحث غير المعلوم على عدم علم المراحل الباقية التي تأتي بعد المرحلة الحالية، وفي هذه الحالة يقوم البحث في كل مرة بتوليد مراحل جديدة ثم يختبر هذه المراحل هل من ضمنها ما يحقق الهدف؟ فإن كان ذلك ينتهي بذلك البحث، وإلا فيكرر البحث بتوليد مراحل جديدة وهكذا.

يعتمد البحث الأعمى على استخدام أدوات خاصة تختلف باختلاف استراتيجيات البحث ، وتقوم هذه الأدوات بفصل المرحلة المحققة للهدف عن المرحلة التي لا تحقق الهدف . سنتناول في هذه الفقرة ست استراتيجيات بحث أعمى.

4-6-1- البحث بالعرض أولاً (Breadth First Search):

تتلخص فكرة البحث بالعرض أولاً أنه في أثناء الانتقال في البحث من جذر الشجرة نتناول المراحل التالية بصورة عرضية، أي نختبر النقاط (Nodes) التي تأتي في المستوى نفسه قبل الانتقال للمستوى الأدنى أي التوسع يكون أفقياً وليس نحو العمق، كما يتضح في الشكل(4-7).

استراتيجية العرض أولاً تعمل وفق منهج (FIFO)، أي الأول دخولاً هو الأول خروجاً (First in First out) وذلك على مبدأ الطابور (Queue).



الشكل(4-7) البحث بالعرض أولاً في شجرة ثنائية

يلاحظ في استراتيجية العرض أولاً ما يأتي:

1. لا تكون مناسبة إلا إذا كانت تكلفة الوصول لكل المراحل متساوية. وهنا يكون السؤال، أي المراحل يتم اختبارها أولاً؟ ولاسيما أننا عندما ننتقل من الجذر تأتي أكثر من مرحلة متفرعة من هذا الجذر وكلها في مستوى عرضي واحد.
2. إذا كان الهدف موجوداً في آخر مرحلة ، يعني أننا نحتاج أن نمر على جميع المراحل.
3. إن رتبة الخوارزمية المستخدمة في هذه الاستراتيجية هي $(O(b^{k+1}))$، لأننا إذا فرضنا مثلاً أن في كل مستو لدينا عدد (b)

من المراحل فإننا في المستوى الأول نحتاج أن نمر على (b) مرحلة، ثم في المستوى الثاني نحتاج أن نمر على مربع هذا العدد لأن كل مرحلة يتفرع منها عدد (b) من المراحل، أي أنه في المرحلة الثانية نحتاج أن نمر على عدد (b^2) وهكذا، إلى أن نصل للمرحلة الأخيرة (k) نكون قد مررنا على (b^k)، وعليه فإن مجموع ما نقوم به من عملية بحث في جميع المراحل يكون ($O(b^{k+1})$).

4-6-2- البحث بحساب التكلفة (Uniform Cost Search):

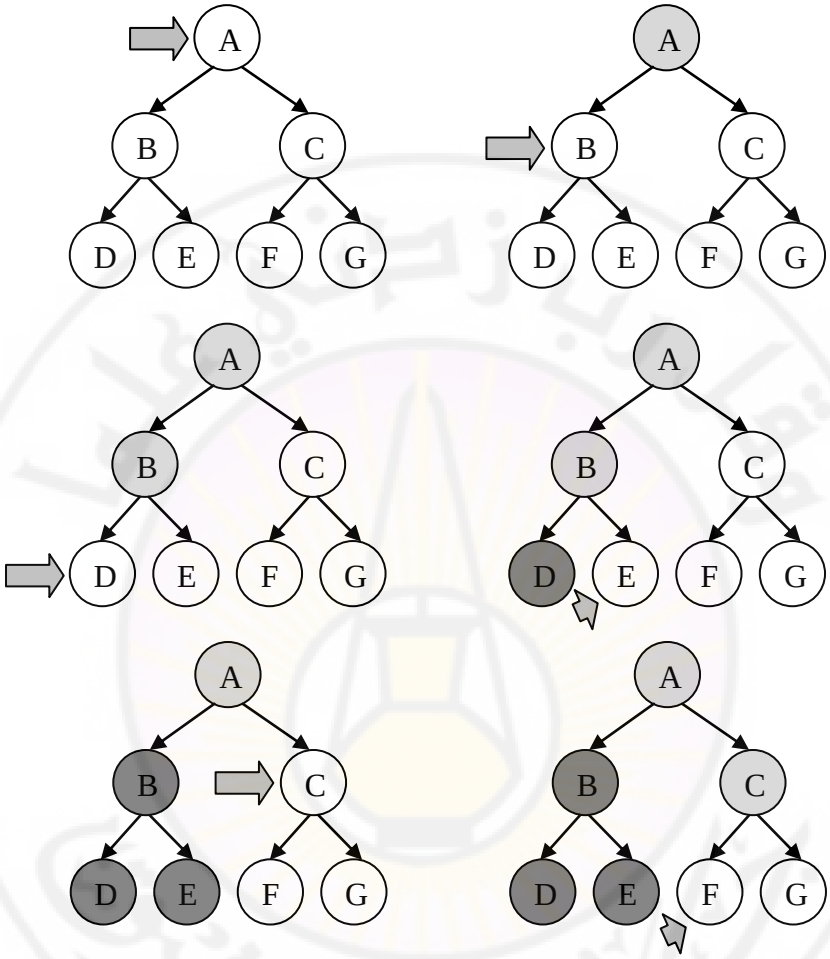
استراتيجية العرض أولاً قد تكون مناسبة إذا كانت تكلفة الوصول لجميع المراحل متساوية. حيث كما لاحظنا فإن استراتيجية العرض لا تهتم بتكلفة الوصول للمرحلة عند الانتقال من مرحلة إلى أخرى . تقوم استراتيجية حساب التكلفة بمنهج استراتيجية العرض أولاً نفسه، ولكنها عندما تنتقل بالبحث من مرحلة إلى أخرى فإنها تأخذ المرحلة التي يكون الانتقال إليها أقل تكلفة، أما إذا كانت جميع المراحل بالكلفة نفسها فإنها تستخدم استراتيجية العرض أولاً.

نلاحظ في هذه الاستراتيجية أنها ربما تكون مثالية إذا كان هنالك تفاوت في تكلفة الوصول من مرحلة إلى أخرى، ولكن يعاب عليها أنها تضيف عمليات جديدة عند اختيار المرحلة التالية مما يؤدي إلى مضاعفة في الزمن المستغرق والمساحة المستخدمة في الذاكرة. فإذا كانت إستراتيجية العرض أولاً من الرتبة ($O(b^{k+1})$) فإن هذه الإستراتيجية تضيف إلى كل عملية انتقال من مرحلة إلى أخرى عمليات اختبار لكل المراحل المتاحة حتى تقف على الأقل تكلفة، ويمكن أن تقدر رتبة هذه الخوارزمية ($O(b^{(C^*/e)})$)، حيث (C^*) هي تكلفة الحل المثالي، و(e) هي تكلفة كل حدث. مع ملاحظة أن (C^*/e) أكبر من ($k+1$).

4-6-3- البحث بالعمق أولاً (Depth First Search):

في البحث بالعمق أولاً (البحث الرأسي) يتم الانتقال للمرحلة التالية في اتجاه العمق، أي تختبر النقاط (Nodes) التي تأتي في الأسفل ، ويستمر التوسع نحو العمق حتى الوصول إلى النقطة الميتة التي لا يوجد أسفل منها نقاط، ثم تتم العودة إلى الخلف لأقرب نقطة في الشجرة يكون فيها تفرع آخر لم يفحص، ويختبر ذلك المسار حتى نهايته، ثم تكرر العملية للوصول إلى نقطة الهدف، وهذا ما يوضحه الشكل(4-8).

إستراتيجية البحث بالعمق يمكن تطبيقها باستخدام تقنية البحث الشجري العادية (Tree-search) والتي تعمل وفق منهج المكدس (Stack) أي الداخل بالأخير يخرج أولاً (Last In First Out)، وتطبق عادة باستخدام دالة عودية (Recursive function). استراتيجية البحث بالعمق لا تحتاج أن تحتفظ بمعلومات عن كل المراحل في الذاكرة، فقط تصطحب معها معلومات الممر الذي يقودها إلى العمق الذي تعمل فيه، وعندما تنتهي من هذا الممر أي تصل إلى العمق فإنها تقوم بحذف بيانات هذا الممر وتعوضها ببيانات ممر جديد ، وهذا مبين في الشكل(4-8)، حيث ظلل الممر أو النقاط المنتهية بلون داكن والمعلومات التي لا زالت على الذاكرة ظللت بالرمادي.



الشكل(4-8) البحث بالعمق على شجرة ثنائية

4-6-4- البحث بالعمق المحدد (Depth Limited Search):

لعلاج مشكلة عدم بلوغ الحل في حالة الشجرات غير محددة العمق والمتفرعة لفروع كبيرة ، أي الشجرات غير المحددة (Unbounded trees)، تستخدم استراتيجية معدلة عن استراتيجية العمق أولاً ، تعرف باستراتيجية العمق المحدد.

في استراتيجية العمق المحدد يتم تحديد عمق أقصى للشجرة ، بحيث لا يتجاوز البحث هذا العمق. فمثلاً، إذا حدد أن العمق يكون للمستوى (L)، فإن كل النقاط في هذا المستوى تعد آخر نقاط في العمق للشجرة. وبهذا تكون مشكلة الممر غير المنتهي ، أي مشكلة الشجيرات غير المحددة قد عولجت. ولكن قد تتولد مشكلة جديدة إذا كان الحل يوجد بعد المستوى الذي حدد أي أن (L<d)، حيث (d) هو المستوى الذي يوجد فيه الحل.

4-6-5- البحث العميق المتكرر (Iterative Deepening Search):

استراتيجية البحث العميق المتكرر ، وتسمى أيضاً استراتيجية العمق أولاً المتكررة تقوم على فكرة تحديد حدود لمستويات العمق تبدأ من (1)، فإذا لم تجد الحل في هذا المستوى من العمق، تنتقل لمستويات عمق أكبر ، وهكذا في كل مرة تؤخذ مستويات أكثر عمقاً حتى الوصول للحل. لذا فإن هذه الاستراتيجية تجمع بين مزايا استراتيجيتي العمق أولاً والعرض أولاً، حيث أنها تصل للنهاية كما تصل استراتيجية العرض أولاً للنهاية، ومع ذلك تستخدم ذاكرة أقل مثل استراتيجية العمق أولاً.

تبدو هذه الاستراتيجية أكثر استهلاكاً للزمن، فالمراحل فيها تتجدد عدة مرات حسب تكرار تحديد العمق. ولكن رغم ذلك فهذه الاستراتيجية ليست مستهلكة للزمن حيث في كل مرحلة تكون أكثر النقاط في المستوى الأسفل أي المستويات الجديدة، وبالتالي لا يكون هناك تأثير يذكر فالتكرار في كل مرة يحدث فقط على المستويات العليا داخل العمق المحدد، رغم أن المستويات الدنيا ستكون مستويات عليا عند الانتقال لعمق جديد وسوف تتكرر مرة أخرى، ولكن كل هذا لا يفرق

كثيراً في الزمن المستغرق في البحث. من هنا نجد أن هذه الاستراتيجية تتزايد بتزايد العمق الكلي للشجرة مع تناقص في عدد الفروع في كل مرة، أي يعني إذا كان لدينا شجرة بعمق (d) فإن هذه الخوارزمية من الرتبة ($O(b^d)$).

4-6-6- البحث ثنائي الاتجاه (Bidirectional Search):

يقوم البحث ثنائي الاتجاه على تشغيل بحثين في وقت واحد ، أحدهما يبدأ من نقطة الانطلاق ، والثاني يبدأ من آخر نقطة ، ويتجه نحو نقطة الانطلاق ، ويتلاقى البحثان في نقطة الوسط. ومن هنا يكون ما كلف من عمل بوساطة البحثين هو ($O(b^{(d/2)}) + O(b^{(d/2)})$)، وهو أفضل من ($O(b^d)$).

يبدو هذا النوع من الاستراتيجيات هو الأفضل، ولكن توجد عدة صعوبات في تنفيذه منها: كيف يمكن تنفيذ البحث الخلفي (Backwards)؟ وما هي النقطة التي نبدأ منها في الخلف؟ وكيف يمكن تتبع الفروع وهي غير معلومة؟. كما قد يكون من الصعب تنفيذ مثل هذا البحث إذا كانت للمشكلة نتائج مختلفة، أو إذا كان الهدف يمكن أن يتحقق بعدة اتجاهات مثل لعبة الشطرنج مما ينتج عنه تعارض في النتائج بسبب تطبيق بحثين في اتجاهين مختلفين.

يقدم الجدول(4-1) مقارنات استراتيجيات البحث الأعمى من خلال أربعة مقاييس تقييم، وهي:

تكملة البحث، والزمن، والذاكرة المستخدمة، ومثالية البحث من عدمه.

التقييم	العرض أولاً	حساب التكلفة	العمق أولاً	العمق المحدد	العميق المتكرر	ثنائي الاتجاه (إذا نفذ)
هل يكتمل البحث؟	نعم	نعم	لا	لا	نعم	نعم
الزمن	$O(b^{(k+1)})$	$O(b^{(C^*/e)})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{(d/2)})$
المساحة (الذاكرة)	$O(b^{(k+1)})$	$O(b^{(C^*/e)})$	O(bm)	O(bl)	O(bd)	$O(b^{(d/2)})$
هل يعد بحثاً مثالياً؟	نعم	نعم	لا	لا	نعم	نعم

الجدول(4-1) مقارنة استراتيجيات البحث الأعمى المختلفة

4-7- البحث التجريبي (Heuristic Search):

يقوم البحث التجريبي أو البحث المعلوم (Informed Search) على معلومات عن المراحل التالية، وأيها قد لا يؤدي للهدف. ففي مثال الشكل(4-1)، والذي استخرج منه شجرة البحث في الشكل(4-2)، نلاحظ أن شجرة البحث استبعدت مراحل مثل (K) و(L) و(M) من البحث لأنها مراحل حسب ما هو معلوم لا تؤدي للهدف الذي هو الوصول إلى (A) انطلاقاً من (J).

4-7-1- البحث بالأفضل أولاً (Best First Search):

تقوم هذه الاستراتيجية على اختيار أفضل النقاط لتكون النقطة التالية. وهنا يأتي السؤال، كيف نعرف النقطة الأفضل؟.

طريقة الأفضل أولاً تستخدم دالة لتقييم النقاط، ومن ثم تحدد هذه الدالة أي النقاط هي الأفضل ، كأن تكون الأفضلية بالتكلفة. هذه الدالة تعرف بالدالة التجريبية (Heuristic Function)، ويرمز لها بالرمز (h). تعتمد هذه الدالة في عملها على المعلومات المتاحة عن المشكلة، فكلما كانت المعلومات المتاحة عن المشكلة أكثر وأدق، كان نتاج هذه الدالة أقرب للدقة في اختيار النقطة الأفضل.

ليست هناك صيغة ثابتة للدالة التجريبية حيث تعتمد هذه الدالة على طبيعة ومعلومات المشكلة المراد حلها، فمن واقع معلومات المشكلة يتم بناء هذه الدالة . وعادة ما يتم تحديد صيغة الدالة بواحدة من طريقتين ، إما بتتبع فكرة المشكلة ومن ثم تحديد صيغة للحل، أو باستخدام طريقة التعلم من الخبرة (Learn From Experience)، وبصفة عامة تستخدم معظم الطرائق معادلة خطية على النحو الآتي:

$$h = C1.X1 + C2.X2$$

حيث (X1) هي الفرص المتاحة للوصول للحل أو المراحل الممكنة للحل أو المسافة للوصول للحل أو خصائص الحل ، حسب طبيعة المشكلة . أما (X2) فهي البدائل الأخرى للحل . ويبقى الثابتان (C1, C2) لتحقيق الملاءمة لبيانات المشكلة.

تعد طريقة الأفضل أولاً شاملة ، وتتدرج تحتها مجموعة من الطرائق مثل، الأفضل أولاً الطماع، وطريقة البحث (A*).

4-7-1-1- الأفضل أولاً الطماع (Greedy Best First):

هذه الطريقة تسمى في بعض المراجع بطريقة الأفضل أولاً ، أي دون إضافة الطماع. تسعى هذه الطريقة إلى تركيز البحث في النقاط التي تكون قريبة من الهدف، مما قد يجعل الوصول للهدف سريعاً. ولتحقيق ذلك تستخدم هذه الطريقة دالة تجريبية بسيطة تعتمد على حساب تكلفة النقطة التالية حتى الهدف (h) وعليه فإن الدالة تكون (h=f).

يمثل الجدول (4-2) بعض مدن الدولة (X) موضح فيه تقدير للمسافة الخطية (Straight Line) لبعدها عن الهدف.

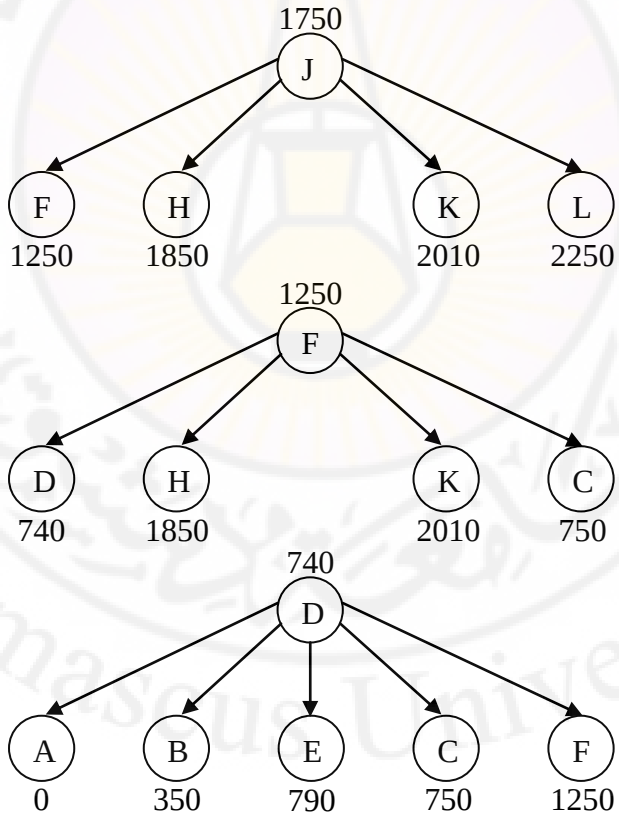
المدينة	المسافة الخطية h_{SLD} (Km)	المدينة	المسافة الخطية h_{SLD} (Km)
J	1750	K	2010
F	1250	L	2250
G	1660	C	750
D	740	E	790
B	350	I	3338
A	0	H	1850

الجدول(4-2) بعض مدن الدولة (X) موضح فيها تقدير للمسافة الخطية

للتعرف على هذه الطريقة ، نعود للمثال في الشكل(4-1)، والخاص بمسألة الوصول إلى (A)، انطلاقاً من (J). فإذا كانت المسافة الخطية من ناحية تقريبية بين (A) وكل مدينة من هذه المدن على النحو الموضح في الجدول(4-2) فإن هذه الطريقة سوف تعمل كما يأتي:

سينتقل البحث من مدينة (J) نقطة الانطلاق ، إلى أقل مدينة من حيث (h_{SLD})، وبشرط أن تكون هذه المدينة ضمن المدن التي تتصل مباشرة بالمدينة (J). وبهذا يكون البحث قد انتقل إلى النقاط القريبة من الهدف.

إذا تتبعنا الخطوط الخارجة من (J) فستكون النقطة التالية هي (F)، حيث تحقق أقل قيمة مقارنة بين القيم التي تحققها بقية المدن التي تلي (J) أي الأقرب للهدف. ثم تصبح نقطة الانطلاق الجديدة هي (F) التي منها ننطلق إلى النقطة الأقرب للهدف والتي هي (D) حيث تحقق (h_{SLD}) يساوي (740). وهكذا حتى نصل (A) كما يتضح من الشكل(4-9).



الشكل(4-9) البحث العميق على شجرة ثنائية

يلاحظ على طريقة الأفضل أولاً الطماع ما يأتي:

1. تركز البحث في النقاط التي تقع في ممر البحث ، ولا تتعامل مع النقاط الأخرى، ما يجعل من خوارزمية هذه الطريقة أقل تكلفة في البحث، ولكنها ليست مثالية، حيث قد تتبع ممراً بتفرعات كثيرة حتى لو وجد ممر مباشر . ففي المثال السابق لو كان هناك مثلاً طريق من (H) إلى (A) مباشرة ، لما اتبعته هذه الخوارزمية رغم أنه الأفضل.
2. اهتمام هذه الطريقة في الحصول على أقل تكلفة قد يؤدي إلى عدم الوصول للهدف. فقد تكون هناك نقطة في نهاية الممر ليست بينها وبين الهدف خط مباشر رغم أنها هي الأقل من حيث (h_{SLD}). وفي هذه الحالة سوف تلجأ خوارزمية البحث إلى العودة للخلف ومعاودة العمل من ممر جديد.
3. إن طريقة الأفضل أولاً تعمل وفق نهج العمق أولاً ، حيث يلاحظ أنها تسلك الممر للهدف بصورة نحو العمق، وهذا قد يؤدي لحدوث مشاكل طريقة العمق أولاً ، وهي عدم الوصول للهدف في حالة الممر ذي النهاية المميتة . ولتجاوز هذه المشكلة فإن هذه الخوارزمية تعود إلى أعلى لتجدد البحث من نقطة ، مع احتمال تكرار النتيجة نفسها، وهذا ما يجعلها غير مثالية.

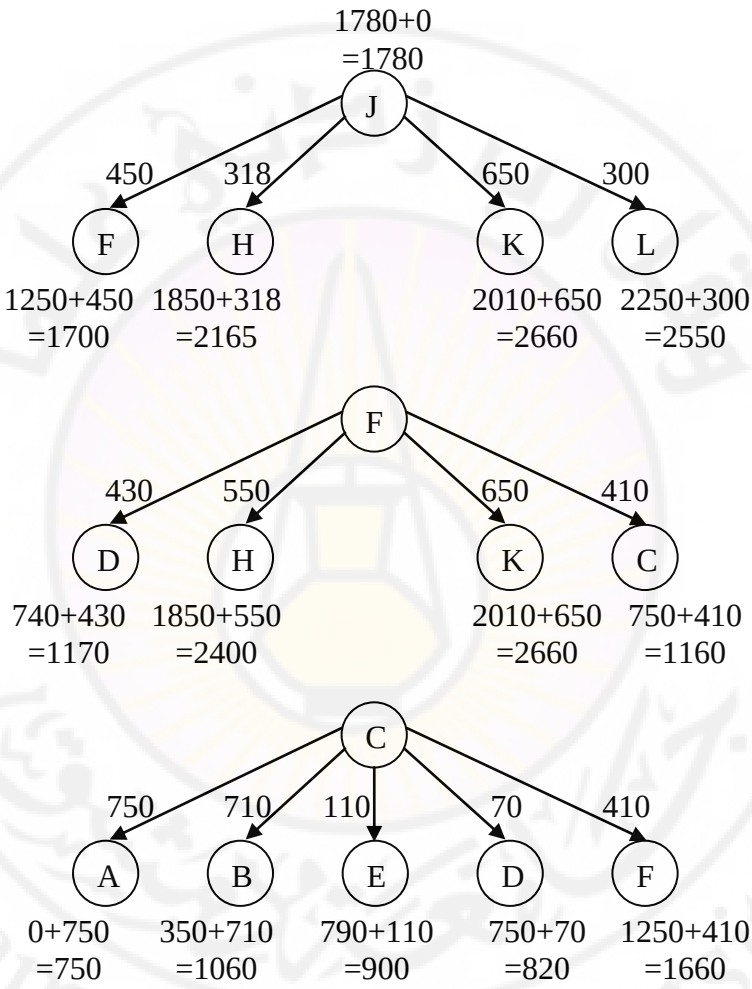
4-7-1-2- طريقة البحث (A*):

تعد هذه الطريقة أكثر الطرائق شهرة ضمن طرائق الأفضل أولاً ، وتعمل وفق الدالة الآتية:

$$f= g+h$$

حيث (g) هي تكلفة الوصول للنقطة، أي طول الممر من نقطة الانطلاق إلى النقطة التالية. بينما (h) هي تكلفة بعد النقطة عن الهدف. وهذا يعني أن هذه الطريقة تتعامل مع النقطة التالية في البحث من خلال حساب المجموع لقيمتين ؛ هما تكلفة الوصول لهذه النقطة، وتكلفة بعد هذه النقطة عن الهدف. من هنا فإن هذه الطريقة تعمل بمنهج الطريقة السابقة ، مع اختلاف القيم التي تحسب قبل الانتقال للنقطة التالية من البحث . لتوضيح ذلك نتناول القيم الموضحة في الشكل(4-1)، والتي تمثل (g) تكلفة الممر للانتقال للنقطة التالية مع بيانات الجدول(4-2).

ومن واقع هذه المعلومات فإن هذه الطريقة تعمل كما هو موضح في الشكل(4-10)، حيث يتم اختيار النقطة التي تحقق أقل قيمة في المجموع لتكون هي النقطة التالية. يلاحظ من المثال أن طريقة البحث (A*) اختارت الانتقال إلى (C) بدلاً من (D) التي انتقلت إليها بالطريقة السابقة.



الشكل(4-10) طريقة البحث (A*)

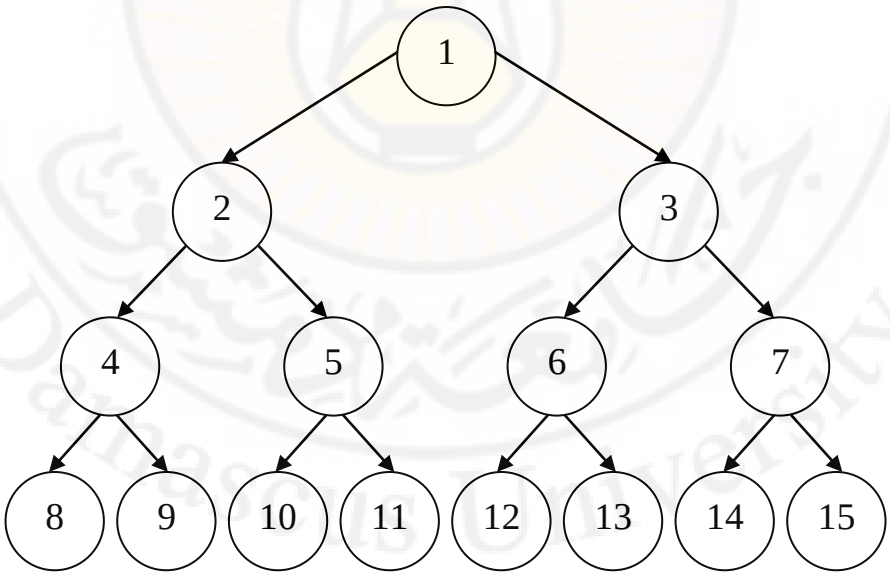
4-8- مثال:

بفرض فضاء حالة بحيث أن الحالة الابتدائية (1)، ودالة النجاح للحالة (n) تعيد قيمتين (2n) و(2n + 1)، المطلوب:

1. رسم فضاء الحالة للحالات من (1) إلى (15).
2. بفرض أن الهدف هو الحالة (11)، حدد ترتيب العقد زيارة العقد باستخدام خوارزمية العرض أولاً، والعمق أولاً، والعمق أولاً المحدود بالمستوى (3)، والعمق أولاً التكراري بالمستوى (3).

الحل:

1. يرسم فضاء الحالة للحالات من (1) إلى (15) كما هو مبين بالشكل(4-11)



الشكل(4-11) فضاء الحالة للمثال

2. يتم زيارة العقد بالشكل الآتي:

العرض أولاً:

(1,2, 3, 4, 5, 6, 7, 8, 9, 10, 11)

العمق أولاً:

(1,2, 4, 8, 9, 5, 10, 11)

العمق أولاً المحدود بالمستوى (3):

(1,2, 4, 8, 9, 5, 10, 11)

العمق أولاً التكراري بالمستوى (3):

(1; 1, 2, 3; 1, 2, 4, 5, 3, 6, 7; 1, 2, 4, 8, 9, 5, 10, 11)

الفصل الخامس

5- نظرية الألعاب (Game Theory):

5-1- مقدمة:

تعرف نظرية الألعاب (نظرية المباراة)، بأنها إحدى وسائل التحليل الرياضي لحالات تضارب المصالح، للوصول إلى أفضل الخيارات الممكنة لاتخاذ القرار، في ظل الظروف المعطاة، لأجل الحصول على النتائج المرغوبة. وهي فرع من علم الرياضيات، كما أنها متداخلة مع علوم الحاسب الآلي، وتم دراستها أيضاً في مجال الاقتصاد، حتى أضحت البنية التحليلية الرئيسية في علم الاقتصاد. فهنا كلمة لعبة تعني أي حالة تتضمن مجموعة من الأطراف حيث أن قرارات طرف ما تؤثر على الأطراف الأخرى بشكل إيجابي أو سلبي، وتهدف النظرية إلى حل ما يعرف باسم "معادلة ناش" بحيث يكون تصرف الطرف المدروس هو الأفضل من حيث النتائج بصرف النظر عن تصرف الأطراف الأخرى.

توجد الألعاب حولنا في كل مكان ، فالسائقون وهم يناورون وسط الزحام المروري يلعبون لعبة قيادة . وتجار الصفقات وهم يقدمون العروض يلعبون لعبة مزادات. وعندما تتفاوض شركة مع إحدى النقابات العمالية على أجور العام المقبل فإنهما تلعبان لعبة تفاوض. وسعر المواد الغذائية في المحلات التجارية تحدده لعبة اقتصادية. فعالم نظرية الألعاب مجال جديد يحلل كيفية ممارسة الألعاب بطريقة عقلانية.

تعتمد حياة الشخص في المجتمعات البشرية بشكل عام على تفاعله مع بقية أفراد المجتمع. تتطلب عمليات الإنتاج والتطور إلى علاقات تعاون على مستوى الأفراد والحضارات، مع وجود احتمال أن تسبب هذه العلاقات كوارث

إنسانية كالصراعات والحروب. فكان السؤال البسيط ظاهرياً، هو كيفية تفاعل البشر معاً لتشكيل علاقات عمل قوية أو ضعيفة؟

بالطبع، العلاقات البشرية معقدة جداً، ولها العديد من المحددات كالمصالح الشخصية، والحب، والعمل، والإكراه. فكانت الإجابة أن قام العلماء بدراسة مجموعة من الأشخاص المستقلين، وتأثير قراراتهم على بقية أفراد المجتمع، أو فئة من المجتمع.

كانت نتائج هذه الدراسة مجموعة من التعاريف لنظرية الألعاب مثل:

- هي دراسة العلاقات بين الأطراف وعوامل التحدي والتفاعل والمساعدة الناتجة عن هذه العلاقة.
- هي علم دراسة المصالح الخاصة بالأفراد، ومدى تأثيرها على المجتمع.
- هي علم دراسة متى تكون الفرصة مناسبة للربح.

بشكل عام نحن نعيش في عالم يسير بانتظام شديد كحركة الكواكب، والذرات، وحتى طريقة هجرة الحيوانات من مكان لآخر. وقد نجح العلماء بفك ألغاز الكثير من الظواهر التي كانت غريبة في الزمن الماضي، لكن هذه الظواهر يمكن القول عنها أنها خاملة، ولا نقصد بكلمة خاملة أنها غير فعالة، إنما على العكس تماماً، فهي تسير ضمن قوانين ثابتة يمكن تمثيلها بمعادلات أصبحت مسلمات، بحيث أن الباحثين يقومون بوضع هذه القوانين دون الحاجة لإثباتها في الأساس، لأن معظم علومنا اعتمد على علوم الذين سبقونا وقاموا بوضع هذه القوانين. فمن المسلمات المعتمدة هي:

- اللاعبون يتصرفون بعقلانية، أي أنهم يحاولون زيادة احتمال وقوع عملية التفوق أو الربح.

- اللاعبون يتصرفون استراتيجياً، أي أنهم يحسبون أو يتكهنون حركة المنافس أو اللاعب الآخر، ويدخلونها في حساباتهم.

لسوء الحظ هناك الكثير من الأنظمة غير الخاضعة لقوانين ثابتة، ولا يمكن توصيفها بقانون، كحركة الكائنات الحية التي تخضع لأكثر من إجابة، فأحياناً تكون حسب رغبة وانطباع الباحث في الموضوع المدروس. كما قد تكون حركة هذه الكائنات الحية لا تخضع إلا لما تراه أو تشعر به من شعور خطر أو شعور بالجوع. حتى موضوع التكاثر بين بعضها، لا يخضع لقوانين يمكن كتابتها أو صياغتها. أيضاً من ناحية الحياة البشرية، لا يوجد أي قانون يمكن أن يصف حالات مثل الحب بين الناس، أو الغيرة، أو الطلاق.

تحاول نظرية الألعاب وضع قانون تفهم عن طريقه العلاقات المشتركة بين المتفاعلين في المجتمع، وكل علاقة بين فردين على الأقل (شخصين، شركتين، سباق بين سيارتين، مدينتين... الخ) تسمى لعبة (Game). فهذه الكلمة في معناها العام مجموعة من القرارات الفكرية والرياضية والعفوية التي يجب اتخاذها لفوز أحد أطراف النزاع في اللعبة، سواء كأشخاص حقيقيين أم اعتباريين. لذلك تسير نظرية الألعاب وفق معادلات احتمالية رياضية بشكل كامل لاتخاذ القرارات التي تكون عند رغبة الشخص، أما تنفيذها أو عدم تنفيذها فيعتمد على طريقة تفكير اللاعب. إذا افترضنا وجود شخص يقوم باتخاذ قرار لوحده، فهذا لا يعني أنه ضمن لعبة، لأن قراره لا يؤثر إلا على نفسه، وليس على طرف ثانٍ.

تدرس نظرية الألعاب إستراتيجية التفاعل عند وجود عملاء متعددين ، لكل منهم تابع المنفعة الذي يعتمد على الحل الأمثل ، أي ربح أكبر مع كلفة أقل . إذا تمكن أحد اللاعبين ، أو أحد الأطراف عموماً، من الفوز ، فإن ذلك يفرض خسارة الآخرين مثل لعبة (Tic–Tac–Toe) أو (X–O) .

يجب التفريق بين الألعاب الاحتمالية التي تركز على الحظ مثل النرد ، والألعاب الإستراتيجية التي لا تعتمد على الحظ مثل الشطرنج (Chess). حيث ينتج ربحاً أو خسارة بالاعتماد على قرارات الخصوم ، وهذا النوع هو الذي تركز عليه نظرية الألعاب.

تساعد نظرية الألعاب في اتخاذ القرار المناسب، فهي تلتقط الحالات الواقعية وتحاكيها من خلال الألعاب ، وتضع لها القواعد ، ثم توجد استراتيجيات تمكن اللاعبين من تحقيق مكاسبهم، وهي تختلف عن تصميم الألعاب حيث يجب التمييز بينهم.

تنطوي نظرية الألعاب على تحليل التفاعلات بين مختلف اللاعبين في الألعاب غير التعاونية، التي لا يتحالف فيها اللاعبون، حيث تركز على التنافس . فالألعاب التنافسية تتطلب بشكل نموذجي لاعبين نسميهما اليمين واليسار ، يلعبان بشكل متناوب بحركات مدروسة . كما يوجد ألغاز تلعب من قبل شخص واحد ، وأخرى لا تحتوي على أي لاعب مثل لعبة كونواي (k). ولكن طبعاً في كل الحالات لا يوجد أي معلومات مخفية أو عشوائية ، حيث أن اللاعبين على معرفة بجميع معطيات اللعبة. الفكرة هنا في اختيار أفضل الإستراتيجيات للتغلب على خصم مثالي.

ليس من الضروري أن يكون اللاعب أو الفرد شخصاً حقيقياً، فمن الممكن أن يكون برنامجاً تطبيقياً مبني على خوارزمية برمجية أو إنسان آلي (Robot).

يمكن تصنيف عدة أنواع لنظرية الألعاب، فبفرض أنه لدينا لاعب يريد اتخاذ قرار، ولكن هذا القرار يؤثر بشكل أساسي على اللاعب الآخر في اللعبة، ويؤثر على اللاعب نفسه، فهذا النوع من الألعاب يسمى (Dilemma

(Prisoner. أما إذا أعطى اللاعب قراراً لم يقتنع به اللاعب الثاني، ولم يهتم بمصالحه، فتسمى هذه اللعبة (Social Dilemma).

تطبق نظرية الألعاب في مجالات واسعة، من أجل فهم الإستراتيجية التي أدت إلى اتخاذ القرار وما هي تأثيراته. كما تتطلب أبحاث نظرية الألعاب دراسة التفاعلات بين الناس وأفكارهم لأنهم هم من يبتكر عدداً كبيراً ومتنوعاً من التقنيات لتحقيق الربح في اللعبة.

بالرغم من ارتباط نظرية الألعاب بالتسالي المعروفة كلعبة الداما (English draughts)، و(X–O)، و(Poker)، إلا أنها تخوض في معضلات أكثر أهمية تتعلق بجميع مجالات الحياة وعلومها كعلم الاجتماع، والاقتصاد، والسياسة، والفلك، والميكانيك، والطب، بالإضافة إلى العلوم العسكرية.

5–2– البدايات:

ساهم العديد من العلماء في وضع الإطار النظري العام لنظرية الألعاب، كعالم الرياضيات الفرنسي (Emile Borel)، الذي كتب أكثر من مقالة عن ألعاب الصدفة، ووضع منهجيات لآلية الألعاب. أما عالم الرياضيات (John Von Neumann)، فقد أسس الإطار الرياضي عبر سلسلة من المقالات امتدت على مدى عشر سنوات (1930 – 1920). وفي عام (1944) ألف (John Von Neumann) و(Oskar Morgensten) كتاباً هاماً أسس لعلم الألعاب بعنوان (The Theory of Games and Economic Behavior). بعد ذلك استعملت أولى النماذج الاقتصادية القائمة على نظرية الألعاب، وقدمت عدة دراسات في العلوم الاقتصادية التجريبية للتأكد من صحة نتائج نظرية الألعاب. كما تم إقحام نظرية الألعاب في علم الأحياء التطورية (Evolution Biology)

حيث ألف (John Maynard Smith) كتاباً بعنوان (Game Theory and the Evolution of Fighting) عام (1972). وفي عام (1994) حصل كل من (John Nash) و(Reinhard Selten) و(John C. Harsanyi) على جائزة نوبل للاقتصاد، لإسهاماتهم في مجال نظرية الألعاب. ونظراً لإنجازات (John Nash) الكبيرة فقد تم تجسيد شخصيته في فيلم سينمائي بعنوان (A Beautiful Mind).

كان (John Nash) في جامعة (Princeton) دائم البحث عن فكرة مبتكرة وعبقرية تغير العالم، فحدثت معه إحدى القصص المشهورة التي أسست لنشوء نظرية الألعاب. ومن أقواله المشهورة، "سابقاً كنا نقول أن أفضل النتائج تأتي عندما يعمل جميع الأشخاص في المجموعة بأفضل ما لديهم لمصلحتهم الشخصية، لكن هذا التعبير خاطئ، إذ يجب القول إن النتائج الأفضل تأتي عندما يقوم كل شخص بعمل أفضل ما لديه لنفسه وللمجموعة وهنا يكون الفوز الحقيقي".

كما تم في الأعوام الأخيرة منح العديد من العلماء أرقى الجوائز، بسبب إنجازاتهم في مجال نظرية الألعاب.

5-3- تعاريف:

فيما يأتي بعضاً من أهم المصطلحات المستخدمة في نظرية الألعاب:

1. اللعبة:

إن مصطلح لعبة في نظرية الألعاب يعني بشكل خاص معضلة أو مشكلة ما، حيث أن عدداً من الأشخاص أو المجموعات (اللاعبون) يشتركون بمجموعة من القواعد والأنظمة تصنع الظروف والأحداث التي تشكل بداية اللعبة. تنظم هذه القواعد الحركات القانونية الممكنة في كل مرحلة من اللعب، ومجموع الحركات أو

الخطوات بمجملها يشكل ماهية اللعبة بالإضافة إلى النتيجة المرغوبة، وهنا يفترض أن اللاعبين أشخاص راشدون يسعون إلى سعادتهم عبر اتخاذهم لسلسلة من القرارات. اللعبة موقف يجب على اللاعبين (على الأقل اثنين) فيه اتخاذ قرار، وكل لاعب يسعى للتنبؤ بأفكار وحركات اللاعب الآخر.

2. الحركة:

في مفهوم نظرية الألعاب فإن الحركة هي التي تنقل اللعبة من مرحلة إلى أخرى، بدءاً من المرحلة الأولى وانتهاء بالمرحلة الأخيرة. والحركة قد تنتقل من لاعب إلى آخر بشكل محدد ومتتابع أو معاً. إن قرار اتخاذ الحركة من الممكن أن يكون ناتجاً عن قرار شخصي أو بالصدفة، وفي الحالة الأخيرة يوجد غرض مثل حجر النرد أو دولاب الحظ، يحدد الحركة المعطاة وفقاً لآلية الاحتمالات.

3. الخرج (النصيب أو النتيجة):

هو مصطلح لنظرية الألعاب يشير إلى ماذا حدث في نهاية اللعبة. في بعض الألعاب مثل الشطرنج (Chess) أو الداما (English draughts) تكون النتيجة واضحة وبسيطة، وذلك بتحديد الخاسر والرابح. في بعض ألعاب الرهان مثل (Poker) يكون النصيب هو النقود، وكمية النقود تحدد بعدد الرهانات التي وضعت في أثناء اللعب.

4. الصيغة الشاملة والصيغة الطبيعية:

يعد البحث في الفرق بين الصيغ الشاملة والصيغ الطبيعية من أهم دراسات نظرية الألعاب. نقول عن اللعبة بأنها في صيغتها الشاملة إذا تم تأليفها وفقاً لقواعد تحدد الحركات الممكنة في كل مرحلة، حيث تحدد على أي من اللاعبين عليه اللعب (الدور)، كما تحدد الاحتمالات الممكنة التي تنتج عن أي حركة للاعب أسندت إليه بالصدفة، كما تحدد هذه القواعد حجم النصيب أو الخرج

الممكن الناتج عن خوض اللعبة. كما أن الافتراض يقول إن كل لاعب لديه مجموعة من التفضيلات عند كل حركة بشكل توقع للخرج الممكن الذي إما سيزيد نصيب اللاعب من الخرج أو يخفضه.

اللعبة في صيغتها الشاملة لا تحتوي فقط على لائحة من القوانين والقواعد التي تحكم تحرك كل لاعب، بل تحتوي أيضاً على مخطط من التفضيلات لكل لاعب، حيث الألعاب الجماعية الشائعة مثل (X–O) أو ألعاب الورق.

إن أبسط الألعاب بصيغتها الشاملة تتضمن كماً هائلاً من المنهجيات والتخطيط، لذلك طوّر الباحثون نمطاً جديداً من الألعاب دعيت بالألعاب بصيغتها الطبيعية، حيث يمكن حساب النتائج بشكل كامل.

تكون اللعبة بصيغتها الطبيعية إذا أمكن وضع جميع النتائج أو الخرج لكل لاعب في حال اتخاذه أي قرار نابع عن إستراتيجية ممكنة اتبعها. وهذا الشكل من الألعاب النظرية يمكن لعبه عن طريق أي مراقب حيادي لا يتأثر بقرارت يتخذها اللاعبون.

5. كاملة المعطيات:

نقول عن اللعبة بأنها كاملة المعطيات إذا كانت جميع الحركات الممكنة معروفة لكل لاعب، كالداما (English draughts)، والشطرنج (Chess)، أما (Poker) فتعد لعبة لا يمتلك فيها اللاعبون إلا قدراً محدوداً من المعطيات في بداية اللعبة.

6. المنهج أو الخطة:

هو قائمة اللاعب بالخيارات المثلى الممكنة في كل مرحلة من مراحل اللعبة. يعد المنهج الذي يأخذ في الحسبان جميع الحركات الممكنة قبل اتخاذ القرار هو منهج لا يخيب، حيث لا مكان للأحداث المفاجئة في هكذا مناهج.

5-4- عناصر اللعبة:

تطلب نظرية الألعاب معرفة عدة نقاط أساسية حتى نتمكن من تطبيقها بنجاح، مثل هوية الأطراف المتفاعلة ضمن اللعبة، وتفضيلات هذه الأطراف، وصفات هذه الأطراف، والتصرفات الإستراتيجية المسموحة لهم، وبالطبع مدى وكيفية تأثير القرارات المختلفة على النتيجة النهائية.

تتألف كل لعبة من ثلاثة عناصر أساسية؛ وهي:

1. اللاعب:

هو متخذ القرار في اللعبة بهدف الوصول إلى النتيجة المطلوبة، مثل مبلغ مادي أو حتى الراحة في المنزل. إذن، فقراراته يجب أن تصب كلياً في مصلحته بصرف النظر عن اللاعبين الآخرين، لذلك يجدر على كل لاعب أن تكون لديه بعض القواعد والقوانين التي يسير عليها حتى يصل لهدفه.

يوجد نوعان من اللاعبين:

- لاعب يفكر بطريقة منطقية.
- لاعب لا يفكر بطريقة منطقية.

فيكون هنا في دراسة الحالة أنه في حال كان اللاعبان يفكران بطريقة منطقية فسيختاران بالتأكيد الخيار الذي يفيد كليهما. أما في حال لم يفكرا بطريقة منطقية، فالإجابة تابعة لطريقة تفكير كل لاعب، وليس من الضروري إعطاء نتيجة جيدة عندها. هذا النوع من اللاعبين هو الأكثر انتشاراً في العالم، إذ إن كل لاعب يعطي قراره بناء على تراكم مشاعره حتى لحظة اتخاذ القرار، ولا يكون لديه تركيز على هدف محدد، أو يقوم بالتركيز على أهداف غير مفيدة.

2. الإستراتيجية:

هو القرار المتخذ من كل لاعب وفق لحالة يجد اللاعب نفسه ضمنها. كما أنها إحدى الخيارات التي يمكن للاعب أن يتخذها، والتي ستؤثر على خيارات اللاعب الآخر. فهي تعبر عن خوارزمية لعب اللعبة. كمثال عليها حالة فتح محل جديد، واتخاذ قرار توسيع العمل. هنا نجد أن هكذا قرار لا يمكن تنفيذه في ليلة وضحاها، لكن يجب اتخاذ العديد من التدابير والأفكار حتى يصل صاحب المحل إلى القدرة على توسيع تجارته، أي أنه قام بالتفكير بخوارزمية معينة للنجاح وقام بالسير بناء عليها.

3. نظام الجوائز:

هو الربح أو الخسارة بعد اتخاذ كل لاعب لإستراتيجية معينة. يهتم نظام الجوائز بما يرغب فيه اللاعب، وليس بما يتوقع اللاعبون الآخرون أن اللاعب يريده. أحياناً تكون هذه الجائزة عبارة عن السعادة التي يكتسبها اللاعب بعد رؤية النتائج التي قام بعمل إستراتيجيته عليها، ومن الممكن أن تكون الجائزة هي حزن هذا اللاعب لفشل إستراتيجيته.

يجب معرفة الجائزة التي يستحقها اللاعب في حال نجاح إستراتيجيته وما هي العقوبة التي يجب تحملها في حال الفشل.

5-5- أهمية نظرية الألعاب (The importance of game theory):

كانت معظم الخطط العسكرية في الحرب العالمية الثانية ضمن مجال نقل الجنود وإيوائهم والدعم اللوجستي، وفي مجال الغواصات، والدفاع الجوي، مرتبطة بشكل مباشر بنظرية الألعاب. بعد ذلك تطورت نظرية الألعاب كثيراً في بيئة علم الاجتماع، ومع ذلك تعد نظرية الألعاب نتاج جوهري من علم الرياضيات.

إن تطبيقات نظرية الألعاب واسعة ومتعددة، وقد أشار مؤلفو النظرية (Von Neumann and Morgenstern) إلى أن الأداة الفعالة لنظرية الألعاب يجب أن ترتبط ارتباطاً وثيقاً بعلم الاقتصاد ونظرية سلوك المستهلك. تعتبر النماذج الاقتصادية وخصوصاً نموذج اقتصاد السوق، سوق المنافسة الكاملة مكاناً مثالياً لاختبار فرضيات نظرية الألعاب. كما أن لنظرية الألعاب استعمالات عديدة في قسم بحوث العمليات، الذي يخوض في مسائل تعظيم الأرباح وتخفيض التكاليف.

كما ترتبط نظرية الألعاب ارتباطاً وثيقاً بعلم الاجتماع وتستخدم على نطاق واسع في السياسة، والعلاقات الدولية.

يرى بعض العلماء أن لنظرية الألعاب مع فيزياء الكم تطبيقات كثيرة لشرح الإدراك البشري وأنماط التفكير غير المنطقي.

5-6- نظرية الألعاب في الذكاء الصنعي:

تعد الألعاب من المسائل الأساسية في الذكاء الصنعي ، ونظرية الألعاب هي خوارزميات واستراتيجيات لحل هذا النوع من مسائل الألعاب مثل (Poker) والبلياردو. فالذكاء الصنعي يعمل على خوارزميات نظرية الألعاب بعرض تطبيقات مهمة واستراتيجيات مناسبة لحلها، ويركز على ثلاث نقاط أساسية:

1. تمثيل اللعبة (game playing):

هي تصميم الطرائق الآلية من أجل الألعاب التنافسية الشائعة بين الناس . تركز على وضع نموذج لطرائق أو أفكار حل الألعاب التقليدية ، ثم تقوم بتوسيع هذه الأفكار للتعامل مع المسائل المشابهة لها ، وتعطي تعليمات للعملاء حول كيفية التعامل مع الخصم.

2. تحديد القرار:

هي مجموعة من القوانين أو الإيضاحات للعملاء، وتستخدم لذلك شجرة القرار في العديد من المسائل من أجل الألعاب التي تلعب بلاعبين ، حيث أن اللاعب الذي يبدأ باللعبة يحاول أن يختار الحل الأكثر مثالية المتاح له . يمكن تمثيل ذلك بما يسمى شجرة اللعبة، التي تتألف من عقد وكل عقدة تمثل اللاعب الحالي، والحالة الحالية للعبة. الاتصالات بين العقد تمثل الانتقالات أو التحركات ضمن قواعد اللعبة. أما الجذر فإنه يمثل اللاعب الابتدائي والحالة الابتدائية ، فإذا كانت حالة عقدة الورقة هي حالة الفوز ، فهذا يدل على فوز أحد اللاعبين في البداية كان أحد اللاعبين يسعى تعظيم أو زيادة النتيجة، بينما اللاعب الخصم فإنه يحاول إنقاصها وهذا ما يسمى إستراتيجية (Minimax).

3. آلية التصميم:

هي تصميم وكتابة الترميز المصدري للبرنامج الذي يعطي القرار النهائي. تعنى أبحاث الذكاء الصنعي بالتفاصيل الدقيقة من أجل إيجاد نموذج قابل للاستخدام في كثير من البيئات الديناميكية.

5-7- تصنيف الألعاب:

عند الحديث عن الألعاب فإن المقصود هي الألعاب بأنواعها المختلفة كالألعاب الإلكترونية أو الرياضية، حيث تدرس فيها نظريات الحركة وبعض النظريات الرياضية والمنطقية. أما نظرية الألعاب فهي تدرس القرارات التي يتخذها الفرد في موقف معين يحدث معه أو تمت مناقشته به. ومع ذلك فهذا لم يمنع من استفادة الألعاب من نظرية الألعاب في عدة مناح توصيفية.

تعرف اللعبة في المعاجم على أنها " تفاعل يقوم به اللاعب ضمن حدود اللعبة، إما لأجل التعلم والخبرة أو لأجل التسلية ". في الواقع فإن نظرية الألعاب قامت بشرح العديد من الحالات ضمن الألعاب، كحالة اللعبة صفرية المجموع التي يوجد فيها شخص رابح وشخص خاسر، وحالة اللعبة غير صفرية المجموع التي تحتمل وجود التعادل.

كان تصميم الألعاب الإلكترونية سابقاً، يعتمد بشكل أساسي على ما يقوم بتصميمه المبرمج، فيكون اللاعب مجبراً على السير في الطريق الذي رسمه المبرمج، وإلا سيكون نصيبه الخسارة وإعادة اللعبة، إما من نقطة حفظ سابقة أو من نقطة البداية. تطورت الألعاب بشكل كبير منذ عام (1980)، مستفيدة من نظرية الألعاب، فأصبح عدد الألعاب كبيراً جداً، وبأنواع عديدة. سوف نورد ستة تصنيفات للألعاب، مع التركيز على أهمها؛ وهي:

1. من ناحية اتخاذ القرار:

هناك ألعاب يتخذ فيها القرار في بداية اللعبة، وأخرى يوجد فيه تغير مستمر في تحديد القرار، وبالتالي يمكن أن تكون الألعاب إحدى القسمين الآتيين:

1) ألعاب ساكنة:

تتميز هذه الألعاب بثباث القواعد وعدد الأطراف، حيث يجب على اللاعبين أن يقوموا جميعهم باختيار استراتيجياتهم في الوقت نفسه. أي أن جميع اللاعبين يتخذون قراراتهم في اللحظة نفسها، ولا يستطيع اللاعب أن يرى أولاً ماذا فعل المنافس ثم يقرر.

2) ألعاب ديناميكية:

تتميز هذه الألعاب بالتغير المستمر في القواعد وأطراف اللعبة، فيمكن للاعبين فيها أن يتخذوا قراراتهم الواحد تلو الآخر، وهي الألعاب الأكثر تفاعلية والأوسع انتشاراً.

2. من ناحية كمية المعلومات:

المعلومات هي عنصر هام من عناصر أي لعبة، حيث أن اللاعب يقوم باتخاذ القرارات بناء على المعلومات المقدمة له، ويمكن أن تقسم إلى قسمين:

1) ألعاب بمعلومات كاملة:

يعرف اللاعبون جميعهم نوايا منافسيهم، ومنافسوهم يعرفون ذلك، وهم يعرفون أن منافسيهم يعلمون ذلك. أي أن الجميع على دراية كاملة بالأهداف التي يريد المنافس أن يصل إليها. إذن، اللاعب لديه معلومات كاملة عن اللعبة إضافة إلى معرفة بالقرارات التي سيختارها اللاعب الآخر، كعملية اقتسام المال بين لاعبين، إذ يقوم لاعب بسؤال اللاعب الثاني إذا كان يريد نسبة أعلى من الربح، فهنا الإجابة معلومة مسبقاً. فتكون هذه اللعبة هي لعبة كاملة مع معلومات كاملة (Perfect Information).

2) ألعاب بمعلومات منقوصة:

في هذا النوع يكون واحد على الأقل من اللاعبين، ليس له علم كامل بنوايا وأهدف منافسيه. إذن، اللاعب لا يملك معلومات كافية، ولا يعلم اللاعب الآخر ما هي الخطوة التالية التي سيقوم بها. لعبة الشطرنج (Chess) مثال جيد عن الألعاب ذات المعلومات الناقصة.

3. من ناحية عدد اللاعبين:

اللاعب هو الركن الأساسي في أي لعبة، وبالتالي يمكن أن تكون الألعاب من ناحية عدد اللاعبين إحدى النوعين الآتيين:

1) لعبة الشخص الواحد أو اللعب الفردية:

هي لعبة فردية لا وجود لتضارب مصالح حقيقية فيها، لأن المصلحة الوحيدة هنا هي مصلحة اللاعب الفردي نفسه. في هذه اللعبة فإن الحظ أو الصدفة هو بنية اللعبة الأساسية وذلك اعتماداً على خلط الأوراق، وعلى ما امتلكه اللاعب من أوراق جيدة وزعت عليه عشوائياً. بالرغم من اهتمام نظرية الاحتمالات بالألعاب الفردية، إلا أنها لا تعتبر من المواضيع المحببة لدى نظرية الألعاب، حيث لا وجود لخصم يقوم باعتماد منهج مستقل ينافس به خيارات اللاعب الآخر، وكمثال عليها لعبة (Solitaire).

2) لعبة الشخصين أو اللعبة الثنائية:

يعتبر نمط الألعاب الثنائية من أكثر الأنماط انتشاراً، ويتضمن العديد من الألعاب المألوفة مثل الشطرنج (Chess) والداما (English draughts)، أو أي لعبة تعتمد على فريقين اثنين. إن المعضلات الأكثر صعوبة هي التي تتضمن عدة لاعبين، كالألعاب الجماعية (Monopoly) و(Poker)، أو أي لعبة تتضمن لاعبين اثنين أو أكثر، حيث تصبح مسألة التخطيط لاختيار الإستراتيجية الأنسب أكثر تعقيداً، بسبب النقص في معرفة كيفية تصرف الخصم، وهذا يستدعي أحياناً التنبؤ بتصرف الخصم بناء على معطيات ومعرفة محددة.

إن الألعاب الثنائية قد تم تحليلها بشكل موسع في نظرية الألعاب، والصعوبة الحقيقية هي في تمديد النتائج التي تم التوصل إليها لتشمل الألعاب متعددة اللاعبين، ففي الألعاب الثنائية تكون جميع الخيارات والحركات الممكنة

بالإضافة للنتائج المتوقعة، أما عندما يكون هناك ثلاثة لاعبين أو أكثر، فإن احتمالات عشوائية معقدة من الخيارات والفرص تنشأ في ظل الظروف لتشكل تعاوناً، أو التحاماً، أو اصطداماً بين اللاعبين.

4. من ناحية الربح:

الربح هو الهدف الحقيقي وراء كل لعبة اقتصادية أو سياسية أو رياضية. تقسم الألعاب من ناحية الربح إلى نوعين؛ هما:

1) ألعاب صفرية المجموع (Zero-Sum Game):

توصف اللعبة بأنها صفرية المجموع، إذا كان المجموع الجبري للأرباح (الخرج) في نهاية اللعبة هو صفر. ويكون في هذه الألعاب مقدار الربح أو احتماله، مساوياً تماماً لمقدار الخسارة أو احتمالها، وهي المرادف لمصطلح تحليل التعادل الاقتصادي، الذي يعبر عن الوصول إلى نقطة اللا ربح ولا خسارة أو لا إنتاج ولا إهتلاك. فهي طريقة ربح أو خسارة يتم لعبها وفق رغبات اللاعب ومطالبه ككسب المال فرضاً، أو قيام الحروب بين الدول، أو قيام شركة بتوسيع مصادرها، أو مجموعة من الحيوانات تتقاتل على بعض الطعام، أو الألعاب الرياضية. في طريقة الربح (Zero-Sum Game) يكون نجاح اللاعب هو خسارة حتمية للاعب الثاني، ففي لعبة (حجرة ورقة مقص) (Rock-paper-scissors)، عند ربح أحد اللاعبين فإنه حتماً سيخسر اللاعب الثاني.

لا يحتمل في هذا النوع فوز كلا اللاعبين معاً، أو خسارتهما معاً. فحتى في لعبة مبارزة بين فارسين، يمكن أن يقوم كلا اللاعبين بضرب بعضهما ما يؤدي لوفاة اللاعبين وخسارتهما معاً، هنا يكون الرابح هو الذي سبق خصمه بالضرب.

عام (1944) أظهر كل من (John Von Neumann)، و (Oskar Morgensten) أن أي (n) شخص في لعبة صفرية المجموع، من الممكن توسيعها إلى (n+1) شخص في لعبة صفرية المجموع. وهكذا فإن ألعاب (n+1) شخص من الممكن تعميمها من الحالة الخاصة للألعاب الثنائية الصفرية المجموع. وإحدى أهم المسائل التي أثيرت في هذا المجال هي أن مبادئ التعظيم والتخفيض تطبق على جميع الألعاب الثنائية الصفرية المجموع، ويعرف هذا المصطلح بمعضلة تخفيض_تعظيم، وقد تم إثباتها عن طريق (Von Neumann) سنة (1928)، ونجح آخرون بالإثبات استناداً لطرائق متعددة.

2) ألعاب غير صفرية المجموع (Non Zero-Sum Game):

تعني أنه بحسب التفاعل بين اللاعبين ستزيد حالة الربح أو الخسارة. فربما خسارة أحد الأطراف، تؤدي إلى خسارة الطرف الثاني. وربح أحد الأطراف، يؤدي إلى ربح الطرف الثاني، وهذا يتضمن حالة التعادل. فعند العمل ضمن شركة، كلما كان عمل الفريق أفضل، كلما انعكس ذلك على نجاح الشركة، والعكس بالعكس. هنا يكون لأطراف أو عناصر اللعبة، الأهداف والنظرات المستقبلية نفسها. ليس من الضروري أن تربح جميع الأطراف أو أن تخسر سوية، فمن الممكن أن يربح طرف ويخسر طرف آخر. كمثال على هذا النوع وجود شركتين، الأولى لصنع وبيع الحلويات، والثانية لصنع وبيع مواد التغليف البلاستيكية. إن نجاح شركة الحلويات يؤدي إلى نجاح شركة مواد التغليف، لوجود تعامل بين الشركتين، لكن خسارة إحدى الشركتين، لا يعني خسارة الثانية، لأنه يمكن للشركة الثانية أن تبحث عن بديل للشركة الأولى. أما في كرة القدم فقد تم ابتكار فكرة النقاط للمباريات

العالمية، كي تتم مقارنة عدد النقاط ببعضها، بصرف النظر عن ربح الفريق أو خسارته، فالنقاط هي التي تحدد الربح والخسارة النهائية.

5. من ناحية عدد الطرق والنهايات:

كلما زاد عدد الطرق وتنوعت النهايات، كلما ازدادت اللعبة إثارة وتشويقاً، وأصبحت أكثر تعقيداً وتشابكاً، سوف نميز ثلاثة أنواع؛ هي:

1) ألعاب ذات طريق واحد ونهاية وحيدة:

كانت جميع الألعاب الأولى ذات طريق واحد ونهاية وحيدة. ففي عام (1958) ظهرت أول لعبة إلكترونية، وهي لعبة (Tennis). كان تصميم اللعبة بسيطاً جداً، وقواعدها ثابتة، لكن قصتها مرسومة بشكل لا يسمح للاعب بأي حدث لم يقم المبرمج برسمه، حيث يؤدي ذلك إلى إنهاء اللعبة وإعادتها من بدايتها، بسبب عدم القدرة على تخزين مجريات اللعبة. لا أهمية في هذه اللعبة لقرارات اللاعب، لأن النتيجة وجميع الخطوات التي يجب على اللاعب السير بها معلومة ومرسومة، لذلك لا يوجد هنا أي أثر لنظرية الألعاب.

2) ألعاب متعددة الطرق بنهاية وحيدة:

يدعى هذا النوع من الألعاب (Role-playing Game)، وهو يعتمد بشكل أساسي على عالم افتراضي ضخم جداً. في هذا العالم تكون القصة الأساسية للعبة تسير، وفيه شخصيات تسير ضمنه، ويوجد ضمن العالم نوعان من المهام؛ هي:

1. المهام الأساسية:

هي المسار الذي قام المبرمج برسمه لإنهاء اللعبة. في هذا النوع من الألعاب، عندما يسير اللاعب ضمن الخريطة، سيواجهه أحد الوحوش المبرمجة. يتناوب اللاعب والخصم في عملية القتال، حيث يبدأ أحدهما بأول ضربة ثم يقوم الخصم بالرد. كلما تقدم اللاعب درجة في اللعبة تزداد قوة الوحوش، مما يزيد من صعوبة القتال. لذلك عند الاستمرار في اللعبة، سيجد اللاعب أن سياسة القتال الوحيدة لكل الشخصيات غير فعالة، لأن بعض الوحوش تصبح قوية لدرجة أنها من الممكن أن تنهي القتال بشكل سريع، فيكون أسلوب القتال معها بهذه الطريقة غير فعال. هذا التصميم للوحش المذكور تم بشكل مقصود حتى يقوم اللاعب بتغيير إستراتيجيته. فمثلاً، يشكل اللاعب فريقاً من ثلاث شخصيات مختلفة. تكون الشخصية الأولى هجومية، والشخصية الثانية لعلاج الشخصية الهجومية، أما الشخصية الثالثة فهي شخصية دفاعية. تزيد هذه الإستراتيجية من فرص الفوز بتكامل ميزات جميع الشخصيات.

نلاحظ هنا أن تعاون جميع أعضاء الفريق ضروري لنجاح الخطة، وهنا تبدأ فكرة نظرية الألعاب، حيث يمكن هزيمة الخصم بأكثر من أسلوب وطريقة، ومعظمها فعال. الفكرة الأساسية هي باختيار الطريقة الأمثل لهذا الخصم، وهنا تكمن فكرة اللعبة التعاونية، أي أن نجاح الفريق كاملاً أفضل من نجاح أحد الأعضاء. كما يمكن لإحدى الشخصيات أن تكون حيادية في اللعبة، ولا يتم استخدامها أبداً، فيكون هذا اللاعب (Free Rider)، وفوز فريقه يؤدي إلى فوزه أيضاً، لكن بدون أي جهد مبذول من طرفه.

2. المهام الجانبية:

تعد مهاماً اختيارية للاعب، فإما أن يقوم اللاعب بتنفيذها، أو لا ينفذها. فهي مجموعة مهمات عشوائية، تم وضعها في الخريطة، تقوم بزيادة قوة اللاعب عند إنهاء كل مهمة، لكنها اختيارية، ويمكن إنهاء اللعبة دون استخدامها. يبقى القرار للاعب فيما إذا كان يحتاج أن ينفذ هذه المهام أو لا.

خلاصة الموضوع، أن الألعاب تطورت لحالات يمكن للاعب أن يكون قراره مؤثراً على اللعبة دون تغيير مجرى اللعبة، إنما تغيير طريقة اللعب.

يستطيع اللاعب في بعض الألعاب من هذا النوع أن يعرف قرار الخصم قبل الشروع باللعب، وبالتالي الاحتياط والتجهز مسبقاً له. وفي بعضها الآخر لا يتم اطلاع اللاعب على قرار الخصم، وهذا يعود لمصمم اللعبة. فاللعبة المشهورة (Pokémon Go) تعتمد على وجود أنواع من المخلوقات اسمها (Pokémon) في عالم افتراضي، وهذه المخلوقات يمكن تصنيفها إلى عدة أنواع؛ هي الناري، والمائي، والصخري، والرعدي، والرملي. يقوم اللاعب بجمع هذه (Pokémon's) لغرض المباراة مع لاعبين آخرين. يكون لكل لاعب حرية اختيار نوع (Pokémon's) للمباراة، فإذا كان اللاعب الأول لديه من كل أنواع (Pokémon's) السابقة، وتحداه أحد اللاعبين، وقال له أنه سيستخدم (Pokémon) من النوع الناري، فالاختيار الأفضل للاعب أن يختار (Pokémon) من نوع مائي، لأنه منطقياً النوع المائي يهزم النوع الناري. هنا علم اللاعب بقرار الخصم، وقام بالوثوق به بشكل كامل، فاختار النوع المائي كي يفوز عليه. تسمى هذه المحاكمة (Nash Equilibrium)، لأن اللاعب أعطى قراره، بناء على ثقته بقرار خصمه.

3) ألعاب متعددة الطرق ومتعددة النهايات:

في هذا النوع من الألعاب، تؤثر القرارات التي يختارها اللاعب على مجريات اللعبة، والتي تؤدي إلى نهايات مختلفة. فبعض القرارات يمكن أن توصل اللاعب إلى نهاية لا يرغب بها، إنما هي نتاج قراراته. فأحياناً عند خسارة اللاعب في إحدى مراحل اللعبة، فإنها تعد نهاية اللعبة، ويتم إعطاء خيار للاعب أن يبدأ لعبة جديدة من البداية. تم تصميم بعض الألعاب بالعديد من النهايات يصل بعضها إلى (27) نهاية مختلفة، وكل نهاية تعتمد على قرارات اللاعب خلال مجريات اللعب.

ينتج عن كل قرار في اللعبة، أحداثاً مختلفة عن الأحداث التي تنتج عن قرار آخر. وهذا ما يعتبر شكلاً ضخماً من أشكال نظرية الألعاب، حيث في بعض النقاط يتم إعطاء اللاعب القرارات التي تقوم اللعبة باختيارها، ويقوم اللاعب بالاختيار بناء على هذه القرارات. من الممكن أن تكون هذه القرارات غير معلومة، وعلى اللاعب أن يتوقع بناء على الموقف الظاهر أمامه. فهي تعتمد على بديهة اللاعب وذكائه، لكن كل قرار يتخذه اللاعب سيغير من نهاية اللعبة.

6. من ناحية التعاون:

تشترك أحياناً عدة أطراف في اللعبة، وهذه الأطراف قد تتعاون وقد تتنافس وقد تنقسم إلى مجموعات متساوية أو مختلفة العدد، لتبدأ اللعبة بينها وفق شروط وأسس متفق عليها. يمكن الحديث عموماً عن نوعين من الألعاب هما:

1) ألعاب تعاونية (Cooperative Game):

هي ألعاب تتكون من عدد من اللاعبين الذين يساعدون بعضهم ويعملون معاً للوصول إلى نتيجة تفيد الجميع، وتساعد في عملية تطور اللعبة، ويمكن أن

تكون بشكل تنافسي لكن مضمونها تعاوني. بفرض أنه في لعبة ما اتفق اللاعبون على بعض الشروط للسير فيها والعمل على أساسها، عندها يمكن القول إن اللعبة أصبحت تعاونية، يربح فيها جميع اللاعبين وليس أحدهم فقط. كمثال واقعي، عند وصول شركة (Apple) للانهيار، قامت شركة (Microsoft) بمساعدتها، وكان الرد الخاص بها هو: "إذا انتهت (Apple)، فإن المنافسة ستنتهي". إذن، هنا اللاعبون عبارة عن شركتين يوجد تنافس بينهما لكن مصالح إحدى الشركات مرتبطة ببقاء الشركة الثانية.

تقوم الألعاب التعاونية على مفهوم معرفة ما هي المحفزات للاعبين المستقلين التي قد تمنعهم من عملية الاندماج والارتباط مع باقي اللاعبين وإنهاء فكرة اللعبة التعاونية لأن الألعاب التعاونية تقوم على مفهومين أساسيين؛ وهما:

- التعاون مابين اللاعبين.
- نظرية المساومة في الألعاب.

تهتم فكرة التعاون على مبدأ ماذا سيربح اللاعبون إذا قاموا بالتعاون، عوضاً عن ربح فرد واحد فقط. حيث في الألعاب التعاونية العادلة يكون ربح كل اللاعبين متساوٍ، أما في الألعاب التعاونية غير العادلة يكون فيها جميع الأفراد رابحين، لكن بقيم ربح مختلفة. يمكن تمثيل ذلك بشركة قام لاعبان باستثمارها حيث دفع اللاعب الأول (70%) من رأس المال والثاني (30%)، عندها يكون لدينا لعبة تعاونية غير عادلة في الربح وبموافقة جميع الأطراف.

نظرية المساومة بين اللاعبين، والتي تعتمد على المخاطرة العادلة بنتائج الربح لكل لاعب اعتماداً على أهداف مشتركة بينهم. وقد تم وضع بعض شروط ضمان المساومة بين اللاعبين حيث تم إيجاد أن المساومة تعتبر من أكثر الطرائق العادلة بين اللاعبين وذلك بوضع المخاطر أمام اللاعبين والمساومة بينهم حتى

يقوم كل لاعب بالحصول على ما يريده تقريباً، لأنه في العالم الحقيقي لا تكون المساومة عادلة، ويمكن في بعض الأوقات استغلال هذه المساومة، على عكس العالم المثالي.

2) ألعاب تنافسية (Non-cooperative Game):

هي الألعاب التي لا تحتمل أكثر من فائز في اللعبة، فتكون غاية كل فرد من اللعبة هو الربح لوحده. يمكن القول إنها ألعاب تعاونية لم يستطع أحد الأطراف فيها من الوصول إلى درجة ثقة بالطرف الآخر، وتكون عندها اللعبة ذات شكل تعاوني لكن مضمونها تنافسي.

حازت الألعاب التنافسية على اهتمام كبير من قبل العلماء، لرغبتهم في معرفة أفضل إستراتيجية ممكنة في هذه الحالات ضمن الألعاب. يجب التنويه إلى وجود ألعاب تنافسية بحتة، وألعاب تعاونية بحتة، وأحياناً ألعاب تتضمن كلا النوعين، كبعض ألعاب الورق، حيث يكون شرط ربح اللاعب مع شريكه في اللعبة، هو أن يصل أحدهما إلى نتيجة معينة، وتكون نتيجة الشريك موجبة. فهنا على اللاعبين محاولة الوصول إلى قيمة ما، مع الانتباه إلى الشريك في اللعبة.

العنصر الأهم في الألعاب التنافسية هو قيمة الربح الناتج من أفعال اللاعب، والإستراتيجية التي تضمن للاعب الوصول إلى النتيجة الأمثل. لكن السؤال الأهم هو كيف سيتصرف اللاعب في الحالات التي تحدث معه ضمن اللعبة؟ وأيضاً على اللاعبين التفكير في قضية أنه كيف من الممكن الوصول إلى أمثل نتيجة، وليست أفضل نتيجة، في كل حالة يتعرض لها اللاعبون. يدعى هذا المصطلح بالتوازن (Equilibrium) وهو من أساسيات نظرية (Nash) التي تدعى (Nash Equilibrium)، ويعرف بأنه اتخاذ القرار الأمثل للاعب بغياب

القوانين المسيرة للعبة، لتبقى اللعبة التنافسية بين لاعبين اثنين أو أكثر متوازنة وتضمن أن كل لاعب يعرف الإستراتيجية التي توصل اللاعب الثاني لمرحلة الموازنة. في حال قام أحد اللاعبين بتغيير إستراتيجيته عندها لن يستفيد اللاعب من هذا التغيير لأن اللاعب الثاني لم يغير إستراتيجيته، فكلا اللاعبين على علم بالإستراتيجية التي يعمل بها اللاعب الثاني، وفي بعض الأحيان تكون النتائج كارثية. أشهر مثال يوضح التوازن هو (Prisoner's Dilemma Game).

يهتم تعريف (Nash Equilibrium) بتصوير حالة مستقرة من الألعاب التنافسية التي يمكن الحفاظ عليها من قبل اللاعبين بدون إعطائهم أي تعليمات. ولا يمكن القول إنها موضوع بسيط، فكل لعبة لها قوانين خاصة تابعة للاعبين وليس كل الأشخاص يملكون تفكيراً متماثلاً مع غيرهم.

يمكن تصنيف أنواع الألعاب التنافسية إلى:

- اللعبة التنافسية الساكنة (Bayes–Nash Equilibrium): هي التي لا يوجد فيها أي معلومات عن اللاعبين.
- اللعبة التنافسية الديناميكية كاملة المعلومات (Sub–game perfect Nash Equilibrium).
- اللعبة التنافسية الديناميكية غير مكتملة المعلومات (Perfect Bayes Nash Equilibrium).

فاللعبة التنافسية الساكنة هي مجموعة الاستراتيجيات التي يقوم بها اللاعب، ويمكن أن تؤدي إلى عدد كبير من الاستراتيجيات اللاحقة، والتي يستطيع اللاعب استخدامها بحرية لاحقاً.

لتشكيل إستراتيجية ديناميكية متغيرة يجب على اللاعب أن يستعلم عن عدد من المتحولات مثل معلومات عن مدى فهم باقي اللاعبين للعبة التي يخوضونها، ومدى خبرة كل لاعب ضمن هذا الموقف الحاصل ولا يمكن القول: إن كل خطوة يقوم بها اللاعب هي إستراتيجية إنما الخطة الكاملة متعددة الخطوات هي الإستراتيجية. تكون الإستراتجية أفضل، كلما كانت المعلومات صحيحة أكثر. فهي عكس الاستراتيجيات الساكنة التي لا تحتاج أي معلومات عن اللاعبين إنما يقوم كل لاعب بحركة بناء على الموقف الحاصل، فتكون عندها كل خطوة هي إستراتيجية بالنسبة له.

تهتم الألعاب التنافسية بقياس النتائج واستنتاج خوارزميات تتوقع نتائج مواقف لاحقة. إن الخوارزميات التي تم دراستها من أجل الألعاب التنافسية تختلف عن خوارزميات الألعاب التعاونية، لأنه في الألعاب التعاونية يتم تغيير قوانين اللعبة بناء على رغبة اللاعبين، إنما في الألعاب التنافسية فتكون القوانين ثابتة.

من مجريات ما تم مناقشته نجد أن الألعاب بدأت بسيطة جداً، كألعاب ليس فيها الكثير من المنطق، وتطورت حتى أصبحت علماً يدرس في الجامعات. فالألعاب تجسيد لخيال مصمم، وذكاء مبرمج، ويمكن فيها تمثيل أمور مستحيلة علمياً في الحياة. فهذا العلم يقوي خيال المصممين ويزيد من قوة التفكير المنطقي للعاملين في مجال الألعاب واللاعبين على حد سواء. فبعض الألعاب تقوم على فكرة قيام اللاعب بتصميم ساحة اللعبة وفق خياراته، ما يزيد من التفكير الإبداعي لدى اللاعبين، حيث تتم مقارنة التصاميم بين اللاعبين عن طريق الشبكة العنكبوتية، والقيام بالتحديات فيها، مما يعطي اللاعب قدرة على وضع قراراته وإجبار بقية اللاعبين على وضع قراراتهم لمواجهة قرارات وتصاميم الخصم حتى الوصول إلى نهاية اللعبة.

5-8- هندسة نظرية الألعاب:

نظراً لقدرة نظرية الألعاب على حل مشاكل متعددة الحالات، وحل المشاكل المعقدة، فقد استطاعت هذه النظرية أن تكون أداة تستخدم في مجال الاقتصاد والسياسة وعلم الاجتماع. تساعد هذه النظرية على فهم كيفية التعامل بين الناس بما يؤثر على قراراتهم الشخصية في ألعاب غير معلومة البيئة ومن الممكن أن لا يعلم أحد الأطراف بنوايا الطرف الآخر، عندما تكون غاية كل طرف الحصول على أكبر ربح ممكن من اللعبة.

كان هذا المصطلح مرفوضاً تماماً من قبل المهندسين بسبب ضعف في التصميم المنهجي للأفكار المطروحة عن النظرية حتى قام العالم (H.S.Tisen) بإصدار كتابه (Engineering Cybernetics)، والذي قام بشرح تطبيق هام جداً عن نظرية (Wiener) الخاصة بالملاحة الجوية والالكترونيات وتقنيات التواصل، وقام بإعطاء تفسير واسع عن طرائق تصميم المتحكمات التي تحتوي على ميزة ضبط محدداتها (Cybernetics) .

نظرية الألعاب الهندسية تقوم بالدلالة على النظريات والمنهجيات التي تقود إلى التصميم الهندسي وعلم التحكم بالمشاكل الممكن حدوثها. فعلى سبيل المثال، من المعروف التعقيد الهائل في شبكات القدرة الكهربائية التي تقوم بتوليد الطاقة ونقلها وتوزيعها عن طريق دارات معقدة. إن تصميم هذه المحطات وبناءها يتم عن طريق اتخاذ قرارات قد تكون في الواقع غير أكيدة في بعض الأوقات، وجيدة في أوقات أخرى. يتم في أيامنا هذه تطبيق طريقة جديدة بدأت في ألمانيا لتوليد الكهرباء عن طريق وضع لوائح توليد طاقة شمسية على سطح كل بناء في المدينة، وربط هذه الأبنية ببعضها البعض، بحيث يمكن توليد طاقة كهربائية نظيفة للمدينة بكاملها دون الحاجة إلى الوقود. لكن هذه الطريقة تحتاج شبكة ذات

محددات معقدة للغاية، وقد تكون في الواقع غير متجانسة، لذلك يجب على العلماء التفكير بطرائق تتغلب على المشاكل التقنية عند مختلف المراحل، وبمعنى آخر يمكن تطوير طريقة تتضمن الأفكار الآتية:

1. إيجاد حلول لجميع المشاكل التي قد تصادف العاملين، أي أن العلماء لديهم معلومات تدل على احتمالية حدوث هكذا أمر.
2. تحليل وتنسيق الاستراتيجيات المترابطة للوصول إلى تحسين للمشاكل عن طريق الاستعانة بصناع القرارات المعقدة للأفكار غير متوقعة الحدوث، بسبب عدم وجود بيانات يمكن التعامل معها والاستدلال بوساطتها على حدوث المشكلة في اللحظة الحالية، لكن يجب أخذ الحيطة وتجهيز القرار إن لم يكن القرار الأمثل. فنظرية الألعاب موجودة، والعلماء يجب عليهم أخذ الاحتياطات لأسوأ حالة ممكنة.

يفكر العلماء هنا وكأنهما في لعبة (Zero-Sum-Game)، والتي تعني أنه إذا فاز الطرف الثاني على اللاعبين وهم العاملون فهذا يعني فشل المشروع. للتقليل من المشاكل يقترح أحياناً بناء نموذج مصغر واقعي عن المشروع لمعرفة المشاكل الممكن حدوثها والعوامل المؤدية لذلك. وبالتالي يمكن فهم نظرية الألعاب الهندسية أنها فهم للمشاكل غير المتوقعة وإيجاد حلول لها.

تدرس وتحلل الألعاب وفقاً لترتيب الحركات ووفقاً لكمية المعلومات وهذا ما يبينه الجدول الآتي:

		Information	
		Complete	Incomplete
Moves	Simultaneous	Strategic (Normal) Form Games with Complete Information	Bayesian Games
	Sequential	Extensive Form Games with Complete Information	Extensive Form Games with Incomplete (Private) Information

يمكن النظر إلى الألعاب من ناحية ترتيب الحركات على أنها ألعاب متزامنة أو تتابعية، أما من ناحية المعلومات في كاملة المعلومات وغير كاملة المعلومات.

إن لعبة استثمار المال (An Investment Game) هي مثال عن الألعاب المتزامنة وكاملة المعلومات، حيث تكون الخيارات محدودة، وتحدد نتائج كل خيار حسب ما يختاره اللاعب الآخر. يحدد اللاعبون خياراتهم في بداية اللعبة دون معرفة مسبقة بما سيختاره المنافس أو اللاعب الآخر.

أما لعبة (Entry Game) فهي مثال عن الألعاب المتتابعة وكاملة المعلومات، حيث تكون الخيارات محدودة، وتحدد نتائج كل خيار حسب ما يختاره اللاعب الآخر. يحدد اللاعبون خياراتهم بشكل متتابع، وبالتالي يعرف اللاعب الثاني ما قام اللاعب الأول باختياره.

يمكن أن تكون لعبة استثمار المال (An Investment Game) مثال عن الألعاب المتزامنة غير كاملة المعلومات، عند عدم معرفة نمط تفكير اللاعب الآخر وتحديد نسبة متوقعة لكل تصرف ممكن له بناء على نمذجة بعض المواصفات التي يتمتع بها اللاعب الآخر، أو معرفة القيمة التقديرية لسلع تعرض في المزاد العلني.

وكمثال عن الألعاب المتتابعة غير كاملة المعلومات يمكن طرح مسألة التوظيف. يقوم طالب الوظيفة بإبداء رغبته في قبوله للوظيفة مع إعطاء وإظهار الميزات التي يتمتع بها، وإخفاء ما يمكن أن يكون ذو أثر سلبي على قبوله. فهنا يحصل المدير على إشارات متتابعة (Signaling) دون معرفة حقيقية بجدارة الموظف الجديد.

لنستعرض كيفية تحليل بعض أنواع الألعاب السابقة:

1. الألعاب المتزامنة (Simultaneous Games):

ترمز ثلاثة عناصر رئيسية في هذه الألعاب هي:

- مجموعة اللاعبين (N).
- مجموعة الأفعال والتصرفات للاعب (i) هي (A_i).
- تابع الربح (Payoff Function) للاعب (i) هو ($u_i : A \rightarrow R$).

في لعبة (Rock–paper–scissors) يكون عدد اللاعبين اثنين وبالتالي مجموعة اللاعبين ($N = 2$). بينما تكون مجموعة الأفعال هي جميع الخيارات الممكنة للاعب أي (Rock, paper, scissors). ويكون فضاء الحل (Outcome Space) ممثلاً بالشكل:

$$A = X_{i \in N} A_i = \{(a_1, a_2, \dots, a_n) : a_i \in A_i , i = 1, 2, \dots, n\}$$

عادة يحلل هذا النوع من الألعاب عندما يكون عدد اللاعبين اثنين باستخدام تشكيل الخيارات المتاحة بشكل مصفوفي (bimatrix)، بحيث يوضع اللاعب الأول ممثلاً لأسطر المصفوفة، واللاعب الثاني ممثلاً لأعمدة المصفوفة.

2. الألعاب التتابعية (Sequential Games):

يقوم كل لاعب بخطوة لعب بناء على خطوة اللاعب الآخر لتنتقل اللعبة من حالة إلى أخرى. لا يمكن استخدام الشكل المصفوفي في تمثيل ودراسة الألعاب التتابعية، بل تستخدم طريقة الأشجار (Game tree)، التي تبين حالة اللعبة في كل لحظة ومسار الخطوات الذي تسبب في الوصول للحالة الحالية، مع اللاعبين الذين اشتركوا في تلك الخطوات.

تتكون الشجرة من عقد ومسارات بين العقد بحيث تمسح جميع الحالات الممكنة. تسمى العقدة الابتدائية بالجذر (Root)، ويوضع بجانب كل عقدة اسم اللاعب المسموح له بتحديد مسار الانتقال. يخرج من كل عقدة جميع الخيارات الممكنة للاعب الحالي على شكل مسارات من العقدة الحالية إلى عقدة أخرى في مستوٍ آخر. كما يكتب على كل مسار الخيار الذي قام باختياره اللاعب. تحدد في المستويات الأخيرة قيمة تابع الربح عندها، وتكون هذه العقد ممثلة لأوراق الشجرة، وليس من الضروري أن تكون بالمستوي نفسه.

5-9- أمثلة عن بعض الألعاب:

لنتعرف على أشهر الألعاب التي تستخدم نظرية الألعاب ولكن في البداية ما هي إستراتيجية اللعبة؟ هي أن اللاعب سوف يختار الطريقة المناسبة، ويحاسب بناء على ذلك.

1. معضلة السجينين (Prisoner's Dilemma).

وهي اللعبة الأكثر شهرة بين نماذج المحاكاة ، وواحدة من أكثر الألغاز المعروفة اليوم، ولها عدة صيغ وسيناريوهات، ولكننا نكتفي بصيغة واحدة. تفترض اللعبة إلقاء الشرطة القبض على عضوين في عصابة أحدهما شاب والثاني فتاة، بتهمة سرقة مصرف معين. الشرطة متأكدة أنهما مذنبان، لكن التهمة ليست ثابتة، لعدم وجود دليل قاطع على أنهما من قاما بعملية السرقة.

تبدأ اللعبة بقيام المحقق بوضع السجينين في مكانين منعزلين والبدء باستجوابهما. يعرض المحقق اتفاقاً على كل مشتبه به بالشكل الآتي:

- إذا لم تعترفا، فستقضيان سنة كاملة في السجن.
- إذا اعترفت على صاحبك، فسيتم إخلاء سبيلك، وحبس صاحبك ثلاث سنوات.
- إذا اعترفتما سوية فستقضيان سنتين في السجن.

يمكن للمحقق إخبارهما بعد ذلك بأن أحدهما قد تكلم فعلاً، والذي يعقد الصفقة أولاً يكون الرابح. يبدأ كل سجين بالانهيار نتيجة عدم قدرته على توقع ما تكلم به الآخر، ورغبته في أن يكون هو صاحب الصفقة وليس الضحية.

ما يسبب معضلة السجينين هو غياب التواصل بينهما، لذلك تكمن المعضلة في الخيار العقلاني . عن طريق النظر إلى الخيارات المتاحة لأحد

أطراف اللعبة، نجد أن خيانة الشريك والاعتراف يؤدي إلى أفضل المكاسب ، وهي إما إطلاق السراح أو السجن لمدة سنتين. في حين أن الخيار اللاعقلاني وهو التزام الصمت ، يؤدي إلى عاقبة أقل سوءاً وهي السجن مدة سنة واحدة . قد يبدو الإنكار هو الحل الأفضل، لكن في جميع الحالات سواء اعترف شريك الجريمة بالأمر أم لا فالاعتراف هو أفضل قرار.

لاحقاً، ظهر ما يسمى النموذج التكراري لمعضلة السجناء ، والذي يقو م على تكرار المواجهة عدة مرات، وبالتالي يكون لدى كل لاعب معرفة بالقرارات السابقة للاعب الآخر. دعا (Robert Axelrod) في عام (1979) إلى إجراء مسابقة حاسوبية لحل النموذج التكراري من هذه المعضلة تحديداً، ودعي الخبراء إلى إرسال حلول مقترحة. قام العديد من الرياضيين والاقتصاديين وعلماء النفس والاجتماع والسياسة باقتراح طرائق لاتخاذ القرار الأمثل عند كل مواجهة. الطريقة التي حققت أفضل النتائج كانت تدعى (Tit–for–Tat)، وهي تقوم على مبدأ بسيط للغاية وهو " تعاون في البداية، وعاقب كل من يخونك بمثل فعله". تبدأ هذه الطريقة بالتعاون، ثم تعيد في كل جولة لاحقة آخر إجراء قام به اللاعب الآخر ، إن تعاون تتعاون، وإن خان تخون.

لتحليل هذه اللعبة نقوم بإدراج الشكل المصفوفي الذي يوضح الحالة المدروسة في الجدول المبين:

Girl / Youth	Cooperate	Defect
Cooperate	2 year 2 year	0 year 3 year
Defect	0 year 3 year	1 year 1 year

يمكن مناقشة عدة حالات وفق نظرية (Nash)، والتي نعلم أن الطرفين يجب عليهما معرفة قرار الشخص الثاني أو توقعه.

- في حال قام الشاب باختيار عدم التعاون مع الشرطة، فهنا تكون الخيارات للفتاة (girl youth) هي إما أن تتعاون وتخرج من السجن ويبقى الشاب مسجوناً، أو أن لا تتعاون ويبقيان سوية لمدة سنة.
- في حال قام الشاب بالتعاون فسيكون للفتاة حالتين إما أن تتعاون ويبقيان سوية لمدة سنتين، أو أن لا تتعاون ويخرج الشاب.

قبل الاستمرار في الحالات سنحلل الوضع الحالي:

- عندما يعترف الشاب فسيكون للفتاة حالتين، إما الاعتراف والسجن سنتين أو عدم الاعتراف والسجن ثلاث سنوات. أي في حال اعتراف الشاب فسيكون هناك ضرر على الفتاة في كل الحالات لذلك اعتراف الشاب ليس من مصلحة الفتاة بأي شكل وسيتوجب عليها الاعتراف عليه حتى يخرجان معاً من السجن.

- في حال لم يعترف الشاب فهنا للفتاة حالتان وهي إما الاعتراف وخروجها من السجن بدون أي فترة سجن، أو عدم الاعتراف والبقاء سنة في السجن، فهي هنا لها مصلحة في أن لا يعترف الشاب.

لكن على أي أساس سيتم الاختيار، ومن سيقوم بخيانة شريكه؟

الموضوع من وجهة نظر الشاب أيضاً هو نفسه من وجهة نظر الفتاة، لذلك لن نكمل دراسة الحالات.

الموضوع كلياً يعتمد على المنطق والثقة بين اللاعبين الشاب والفتاة، وهو بالضبط فكرة الألعاب التنافسية والألعاب الاعتمادية. بمعنى آخر، إذا كان الشاب والفتاة يثقان ببعضهما، وهي لعبة اعتمادية، فلن يعترفا على بعضهما، وسيتم سجنهما سنة، وهي الحالة الأفضل. لكن في حال عدم الثقة، سيكون لدينا لعبة تنافسية، وسيكون لدينا أحد الشخصين أو كلاهما في السجن اعتماداً على فكرة من يثق بالثاني أكثر.

مع أن عدم الاعتراف من قبل كل منهما يعطي النتيجة الأفضل بمجموع سنتين فقط من السجن . إلا أن اعتراف الشريك فقط يعني مخاطرة كبيرة تؤدي للسجن ثلاث سنوات.

في هذه الحالة ، الاعتراف يبدو غريباً لكنه الحل الأفضل للمصلحة الشخصية وبصرف النظر عن إجابة الشريك. ففي حال اعتراف أحدهما وعدم اعتراف الشريك، فإنه لن يسجن، وهو أفضل من حكم سنة واحدة لو لم يعترفا كلاهما. وفي حال اعتراف أحدهما واعتراف الشريك فسيكون الحكم سنتين لكل منهما، وهو أفضل من حكم ثلاث سنوات في حال عدم اعترافه واعتراف شريكه.

أي أنك دائماً أفضل حالاً في حال اعترافك وهذه هي النتيجة الأفضل لنظرية اللعبة في حالة كهذه، ومع كونك لا يمكنك الثقة بشريكك كونه سيستفيد من التخلي عنك، فالأفضل أن تتابع مصلحتك الشخصية فقط.

ولكن في حالة المحبة والإيثار، فالحل الأفضل هو عدم الاعتراف لأن ذلك يؤدي إلى السجن مدة ثلاث سنوات وإخلاء سبيل الشريك، وهذا ما يتوقعه الشخص. ولكن عند تبادل المحبة فإن الشريكين سوف يسجنان سنة واحدة فقط وهو الحل المثالي.

في عالمنا الحقيقي لا تطبق هذه القوانين، لأن الناس ليس لديهم تفكير يساعد الأطراف كلها، إنما لديهم تفكير أناني، مما يؤدي أن العالم بمعظمه يعيش حياة لعبة تنافسية وليس اعتمادية. لذلك قام الدكتور (Nash) بتطوير مجموعة قواعد أعطاها اسم (Equilibrium Concepts)، والتي تعبر عن إعطاء كل لاعب أفضل الخيارات الممكنة له (Best Response)، بناء على قرارات اللاعب الآخر.

يجب الأخذ بعين الاعتبار خيارات خيانة أحد الأطراف، ويجب على الطرفين دائماً الحذر من تغيير المخططات، كما في لعبة الشطرنج (Chess)، إذ إن اللاعب من الممكن أن يغير طريقته كاملة بسبب تغيير أسلوب لعب اللاعب الذي يلعب ضده، وتتغير كل احتمالات اللعب عنده. تسمى هذه الإستراتيجية (Minimax Strategy)، وهي قاعدة اتخاذ قرار تعبر عن ضرورة تقليل الخسائر في أسوأ الحالات وبتعبير آخر، إستراتيجية (Minimax) هي أصغر قيمة للاعب يقوم اللاعب الثاني بإجبار اللاعب الأول على اختيارها (أي الخيار الأسوأ الذي من الممكن أن يقوم به اللاعب الثاني)، ويتم اختيار هذه الطريقة فقط في حال كان اللاعب لا يستطيع الاعتماد على اللاعب الآخر في أثناء اللعبة. أما عكس

هذه الإستراتيجية فهي (Maxmin Strategy)، والتي تعبر عن أن اللاعب سيختار أفضل حالة ممكنة له، وإن كانت ستؤدي به إلى مشكلة معينة. أي بالعودة للمثال السابق إذا تم إعطاء أحد اللاعبين مبلغ مالي ضخم حتى لا يعترف على اللاعب الآخر عندها في حال الطمع سيختار هذا القرار، مما يؤدي إلى سجنه ثلاث سنوات، مقارنة باللاعب الآخر الذي سيخرج من السجن بدون عقوبة. لكن خوارزمية (Minimax) هي الأكثر منطقية.

عند استخدام نظرية (Nash) للتوازن فإننا نستخدم عبارة " اتخاذ القرار الأمثل للاعب بغياب القوانين المسيرة للعب ".

لتوضيح المقصود بهذه العبارة نفرض أنه لدينا شارع فيه تقاطع وقد صادف في هذا الشارع مرور سيارتان متعاكستان. كانت إشارة المرور خضراء أمام إحدى السيارات وحمراء أمام السيارة الثانية. التصرف المنطقي هو أن تسير السيارة ذات الإشارة الخضراء وتنتظر السيارة الثانية حتى يحين دورها، لكن بفرض شرطي المرور لم يكن موجوداً، وهنا الرقابة على الشارع تعتمد على ثقة الناس ببعضها فنجد في هذه الحالة أربع حالات للدراسة:

- تسير السيارة ذات الإشارة الخضراء، وتتوقف السيارة ذات الإشارة الحمراء، وهو تصرف منطقي.
- تتوقف السيارة ذات الإشارة الخضراء، وتسير السيارة ذات الإشارة الحمراء، وهو تصرف خاطئ لكنه ممكن.
- تتوقف السيارتان معاً، وهو موقف غريب ومحرج وليس صحيحاً.
- تسير السيارتان معاً وتصطدمان، وهو الخيار الأكثر ضرراً.

السيارتان لهما الحرية باختيار القرار الذي يريحهما، لكن سيقوم الاثنان باختيار الخيار الأول، لأنه أكثر أمناً ومنطقية من الخيارات الباقية. فحتى بغياب

شرطي المرور وكون قرار سير السيارتين معاً بالوقت نفسه سيكون أفضل من الناحية الفردية، لكن القرار الآمن هو أن يعملان وفقاً للقانون بغياب مسير القانون وهو شرطي المرور.

وحتى الآن قمنا بفهم أن نظرية الألعاب هي دراسة القرارات لكل لاعب على حدة ومعرفة تأثير هذا القرار على اللاعبين الآخرين، لكن في الواقع إذا نظرنا إلى العالم حولنا سنجد أن يوجد تعاون في الكثير من الحالات بين اللاعبين، ونجد حالات من عدم التعاون بينهم في الواقع.

لكن يجب علينا فهم أن نظرية الألعاب أيضا تهتم بالتعاون بين اللاعبين وليس فقط مدى التحدي بينهم (أي الفرق بين اللعبة التنافسية والتعاونية) مع أخذ بعين الاعتبار أن الخوارزميات التي تم دراستها من أجل الألعاب التنافسية تختلف عن الألعاب التعاونية، لأن في الألعاب التعاونية يتم تغيير قوانين اللعبة بناء على رغبة اللاعبين إنما في الألعاب التنافسية كانت القوانين هي من سيرت اللاعبين وعلى أساسهما قام كلاهما باختيار القرار (مثل في مثال السجن)، عندها يكون السؤال الأهم: متى يجب أن يكون لدينا تعاون في اللعبة، ومتى يجب أن يوجد تنافس؟

2. لعبة المتجرين المتنافسين (Contestable Market):

نفترض اللعبة وجود شركتين أو متجرين يبيعان منتجاً متشابهاً نوعاً ما. يبيع كلا المتجرين في الحالة الابتدائية أكواباً بسعر (10 S.P) للكوب الواحد ، محققين ربحاً بمقدار (4 S.P) لكل كوب ، وبمتوسط أرباح يومية (400 S.P) ناتجة عن بيع (100) كوب. هنا المتجران لديهما واحد من خيارين أساسيين، هما الحفاظ على السعر، أو تخفيض السعر لتكبير الحصة السوقية.

في حال قام المتجر الأول بخفض السعر من (10 S.P) إلى (9 S.P)، فهو بذلك سيجذب المزيد من الزبائن محققاً أرباحاً بمقدار (450 S.P) يومياً، من (150) زبون. بينما المتجر الثاني سيربح (200 S.P) فقط من بقية الزبائن. هنا سيجد المتجر الثاني نفسه مجبراً إما على الاستمرار بالسعر العادي والخسارة، أو تخفيض السعر ليستمر بالمنافسة ويربح بذلك (300 S.P) يومياً.

تطبق حالة تخفيض (تكسير) الأسعار عادة بشكل متلاحق إلى أن يصل المتجران إلى هامش ربح صغير جداً ، ويكونان في حالة توازن ثابت مع أرباح بالحد الأدنى.

يبين الجدول مصفوفة الحالات للمسألة:

المتجر الثاني / المتجر الأول	الحفاظ على السعر	تخفيض السعر
الحفاظ على السعر	400 S.P 400 S.P	450 S.P 200 S.P
تخفيض السعر	200 S.P 450 S.P	300 S.P 300 S.P

الحل الأفضل عموماً هو تثبيت السعر، لكن بوجود التنافسية فالقرار الأنسب هو تخفيضه.

بالنظر إلى جميع الاحتمالات، نجد أن أفضل خيار للمتجرين هو الحفاظ على السعر حيث يحققان ربحاً عالياً لكل منهما . بينما تخفيض السعر من قبلهما معاً هو الحل الأسوأ ، حيث أن الربح الإجمالي هو الأدنى . بالنسبة لكل من المتجرين الأمر معقد، لأن كل لاعب يستطيع معرفة تصرف منافسه ، وبناء تصرفه عليه. لذا ومع أن الحل الأفضل لكل متجر هو تخفيض السعر على أي

حال، فهذا الحل ظاهري فقط كونه يسمح بمنحدر مستمر للأسعار يؤذي المتجران معاً، مما يجعل الحل الأفضل هو تثبيت السعر.

بالطبع تبقى المشكلة دائماً هي مقدار الثقة بين الطرفين، فكل من الطرفين قد يحاول خداع الآخر وسرقة زبائنه بتخفيض السعر، وللحد من ذلك يجب عقد اتفاق بينهما لتثبيت السعر، أو حتى رفعه بحيث يزيدان أرباحهما قدر الإمكان، لكن هذا العمل ليس فقط غير أخلاقي، بل يعد جريمة كبيرة في معظم البلدان كونه يعتبر مؤامرة احتكارية وعادة ما يعامل بقسوة من قبل القوانين والحكومات، لكن كون هذا النوع من التعاون محرماً قانونياً لا يعني أنه لا يستطيع التهرب من الأمر في بعض الحالات وهنا نستعرض مثالين:

- منظمة الدول المصدرة للنفط (OPEC):

رغم أنها منظمة دولية كبرى، إلا أنه إذا اتبعت ممارساتها من قبل الشركات في أي دولة لاعتبرت الاتفاقية غير قانونية أبداً. فالمنظمة المكونة من العديد من الدول العربية بالإضافة لبعض الدول الأخرى ، تنتج نسبة كبيرة من الإنتاج النفطي العالمي ، ما يجعلها شبه محتكرة للنفط وقادرة على فرض سعرها إلى الحد الأقصى لتحقيق أكبر أرباح ممكنة.

قد تفكر البلدان الأعضاء بالانفصال ورفع إنتاجها للحد الأقصى لتحقيق أرباح أكبر، لكن ذلك سيؤدي لتتابع الحال من عدة بلدان، وتحويل سوق النفط إلى سوق حرة بأسعار رخيصة، تؤدي بالنتيجة إلى خسارة جميع المصدرين.

- شركة (De Beers) المحتكرة للألماس:

تحتكر شركة (De Beers) سوق الألماس العالمي، والأمر هنا لا يأتي مصادفة فالشركة أتت أصلاً من اتحاد العديد من الشركات المنتجة والبائعة للألماس لتكوين شركة كبيرة تشبه (OPEC) من حيث الآلية ، وتقوم بالحد من

العرض للتحكم بسعر الألماس ورفعه إلى حد بعيد . فرغم الكمية الكبيرة الموجودة في العالم إلا أن هذا الحجر الكريم باهظ الثمن جداً ، لأن هذه الشركة تتحكم بالكمية المباعة منه بطريقة رفعت أسعاره لأضعاف عديدة، وحولته اليوم إلى واحد من أغلى المجوهرات في العالم ، بينما آلاف الأطنان منه مخبئة في أقبية وخزائن غير مخصصة للبيع.

3. لعبة استثمار المال (An Investment Game):

سوف نبدأ بمناقشة حالة أولى لا يمكن إدراجها ضمن نظرية الألعاب، لننتقل بعد ذلك إلى حالة ثانية تعالج وفق نظرية الألعاب.

الحالة الأولى:

بفرض أن شخصاً يدعى أسامة يريد استثمار (100 $)، ولديه إحدى خيارين، فإما أن يضع المبلغ بفائدة مضمونة مقدارها (10%)، بصرف النظر عن نجاح المشاريع الاستثمارية أو فشلها، أو أن يضع المبلغ بشكل استثماري عقاري يقبل حالة الربح أو عدمه، حسب نجاح المشاريع الاستثمارية أو فشلها. يحصل المودع على ربح مقداره (20%) في حال نجاح المشروع، ولكنه لا يحصل على أي ربح في حال عدم نجاح المشروع بالشكل المطلوب.

يبين الجدول مصفوفة الحالات للمثال:

الحالة / الاستثمار	نجاح	فشل
فائدة مضمونة	10%	10%
استثمار عقاري	20%	0%

بفرض أن احتمال نجاح المشروع هو (p)، فيكون احتمال فشل المشروع هو ($1-p$). إذن، احتمال أن يصبح رصيد أسامة ($ 120) هو (p)، واحتمال أن يبقى رصيده ($ 100) هو ($1-p$). وبالتالي يمكن أن يكون الرصيد في نهاية السنة المالية هو:

$$120.p + 100.(1 - p) = 100 + 20.p$$

فإذا كان الاحتمال ($p = 0.5$)، فإنه من المتوقع أن يصبح الرصيد ($ 110). وعموماً، فمجرد أن يكون ($p > 0.5$)، يصبح الاستثمار العقاري هو الأفضل.

من الواضح أن قرار أسامة لا يعتمد على بقية المستثمرين، كما أنه غير مؤثر على مقدار الربح في الشركة. وبالتالي لا يمكن دراسة هذه الحالة ضمن نظرية الألعاب، إلا إذا أضيفت علاقات وقيود للمسألة، كأن نقول إنه كلما زاد عدد المستثمرين، كلما زادت فرص الربح.

الحالة الثانية:

هذه الحالة مطابقة للحالة الأولى، ولكن نجاح المشروع العقاري مرتبط بكون رأس مال المشروع يساوي ($ 200)، وعندها يصبح رصيد المستثمرين في نهاية العام يساوي (120%). لنفرض وجود شخص آخر يدعى ملهم، يريد استثمار ($ 100)، وهو بحالة تفكير مشابهة لحالة أسامة، مع عدم وجود تواصل بينهما، فكل واحد سوف يقرر لوحده، ولكن قراره سيكون مؤثراً على مقدار ربحه وربح المستثمر الآخر.

يبين الجدول مصفوفة الحالات المحتملة:

ملهم / أسامة	فائدة مضمونة	استثمار عقاري
فائدة مضمونة	10% 10%	0% 10%
استثمار عقاري	10% 0%	20% 20%

يبين الجدول أنه إذا علم أسامة بأن ملهم سوف يختار الفائدة المضمونة، فمن الطبيعي أن يختار هو أيضاً الفائدة المضمونة. وإذا كان قرار ملهم الاستثمار العقاري، فمن الأفضل أن يختار هو أيضاً الاستثمار العقاري.

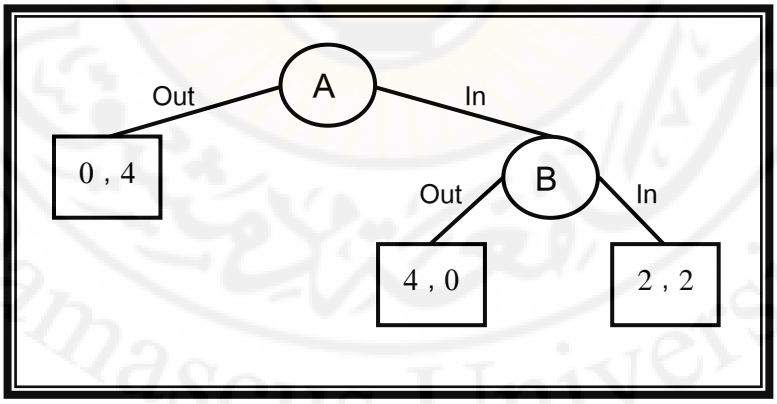
كي يعطي أحدهما القرار الأفضل، فمن المفروض معرفة سلوك الطرف الآخر ودراسة كيفية اتخاذه للقرار والظروف المؤثرة على ذلك. وأيضاً يجب دراسة فيما إذا كان الطرف الآخر يدرس كيفية تصرف الطرف الأول.

4. لعبة الجبان (Game of Chicken):

عرضت هذه اللعبة ضمن فلم سينمائي عام (1955)، حيث تتحدث عن وجود شخصين يتنافسان للتقرب من إحدى الفتيات. في نهاية القصة يتواجد الشخصان ضمن مركبة مسرعة باتجاه أحد المنحدرات الصخرية. يفوز بالامتحان الشخص الذي يبقى بعد منافسه، لأنه سيكون أكثر شجاعة، بينما يخسر الامتحان الشخص الذي يقفز من المركبة لأنه سيكون جباناً. يوجد عدة خيارات، فإذا تأخر المتنافسان للحظة الأخيرة فإنهما سيخسران حياتهما، وبالتالي سيخسران قلب الفتاة أيضاً، فثمن الشجاعة هنا هو الموت الشريف. أما إذا جبن المتنافسان وقفزا سوية، فإنهما سيربحان حياتهما مقابل خسارة قلب الفتاة. أما إذا قام أحدهما بالقفز أولاً فإنه سوف يربح حياته، مع إعطاء فرصة للآخر كي ينجو بنفسه وبقلب الفتاة. إذا

علم أي منهما بما سيقوم به المتنافس الآخر فإن قراره سيكون صائباً، لكن لا يستطيع أحدهما انتظار قرار منافسه كي يتخذ قراره. بمعنى آخر، فإن اللاعبين يجب أن يتخذا قراراتهما بشكل متزامن.

أما في ألعاب أخرى فإن قرار اللاعب الأول يكون بناءً على قرار اللاعب الثاني. فعلى سبيل المثال في لعبة (Entry Game)، إذا أرادت الشركة (A) التي تنتج نوعاً معيناً من العصائر الدخول إلى سوق جديد محتكر من قبل الشركة (B)، عندها فإن قرار الشركة (B) سيكون بناءً على ما ستقرره الشركة (A). فإذا قررت الشركة (A) عدم المنافسة، فإن اللعبة تنتهي دون أي قرار جديد للشركة (B). أما إذا قررت الشركة (A) المنافسة، فإن أمام الشركة (B) خيارين، إما المنافسة سواء بالسعر أم القيام بحملة دعائية، وإما الخروج من المنافسة وخسارة السوق. في هكذا نوع من الألعاب نستخدم تمثيلاً شجرياً يوضح الخيارات الممكنة وهذا ما يوضحه الشكل(5-1)، مع وضع كيفية تقاسم السوق بين أطراف اللعبة في عقد النهاية.



الشكل(5-1) التمثيل الشجري للعبة (Entry Game)

5. المساومة (Bargaining Game):

يعد السوق مثالاً جيداً لنظرية الألعاب، فمنذ لحظة دخولك للمحل متفحصاً البضاعة تكون قد بدأت اللعبة. المستهلك يريد أرخص سعر وأعلى جودة، والبائع يريد البيع بأعلى سعر، والتخلص من البضائع الرديئة الجودة. عندما تبدأ المساومة، والجدال حول السعر، تكون قد وصلت اللعبة لذروتها، والرابح هو الذي يستطيع توقع حركات الآخر، فعندما يتوقع المستهلك بأنه إذا خرج من المحل دون الشراء سيجري وراءه البائع فسيكون هو الرابح إن صح توقعه وخاسر إن لم يصح. سوف نستعرض مثالاً بسيطاً عن عملية المساومة دون الأخذ بعين الاعتبار بعض المحددات الواقعية.

لنفرض وجود بضاعة عند أحد الباعة (Seller) بكلفة (v_s = 50 S.P). يريد البائع بيع بضاعته للمشتري (Buyer) بمبلغ أعلى وليكن (v_b)، وبالتالي يكون:

$$v_b > v_s > 0$$

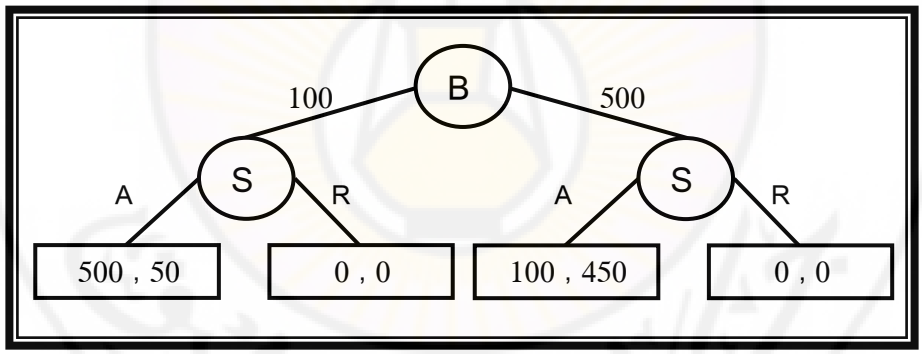
بالنسبة للمشتري فإنه يرغب بدفع (v_s)، بينما يرغب البائع بالحصول على (v_b)، لتبدأ عندها لعبة المساومة بينهما. من الطبيعي أن تكون المساومة باقتراح من قبل أي منهما بين قيمتي (v_s) و(v_b). نشير للاقتراح المقدم من قبل البائع بالرمز (p_s)، ونشير للاقتراح المقدم من قبل المشتري بالرمز (p_b)، ومن الطبيعي أنه في حال كان المبلغ المقترح من قبل المشتري (p_b) أعلى من المبلغ المقترح من قبل البائع (p_s)، عندها فإن عملية البيع والشراء ستتم بنجاح. يمكن نمذجة اللعبة كما يأتي:

$$u_b(p_b, p_s) = \begin{cases} v_b - p_b, & if\ p_b > p_s \\ 0, & otherwise \end{cases}$$

and

$$u_s(p_b, p_s) = \begin{cases} p_b - v_s, & if\ p_b > p_s \\ 0, & otherwise \end{cases}$$

بفرض أن المشتري اقترح مبلغ (500 S.P) أو مبلغ (100 S.P)، بينما البائع يعرض بضاعته بمبلغ (v_b = 600 S.P). وبما أن كلفة البضاعة على البائع هي (v_s = 50 S.P)، فإنه أمام خيارين هما، إما قبول العرض (Accepting) أو رفض العرض (Rejecting). يمكن الاستعانة بالتمثيل الشجري لتوضيح مسألة المساومة، وهذا ما يبينه الشكل(5−2).



الشكل(5−2) التمثيل الشجري للعبة (Bargaining)

فإذا كان اقتراح المشتري مبلغ (100 S.P)، فإنه:

$$u_b(100, p_s) = \begin{cases} 600 - 100, & if\ 100 > p_s \\ 0, & otherwise \end{cases}$$

and

$$u_s(100, p_s) = \begin{cases} 100 - 50, & if\ 100 > p_s \\ 0, & otherwise \end{cases}$$

وبالتالي نحصل على الثنائية:

$$(u_b, u_s) = (500, 50): Accepting$$
$$(u_b, u_s) = (0, 0): Rejecting$$

أما إذا كان اقتراح المشتري مبلغ (500 S.P)، فإنه:

$$u_b(500, p_s) = \begin{cases} 600 - 500, & if\ 500 > p_s \\ 0, & otherwise \end{cases}$$

$$and$$

$$u_s(500, p_s) = \begin{cases} 500 - 50, & if\ 500 > p_s \\ 0, & otherwise \end{cases}$$

وبالتالي نحصل على الثنائية:

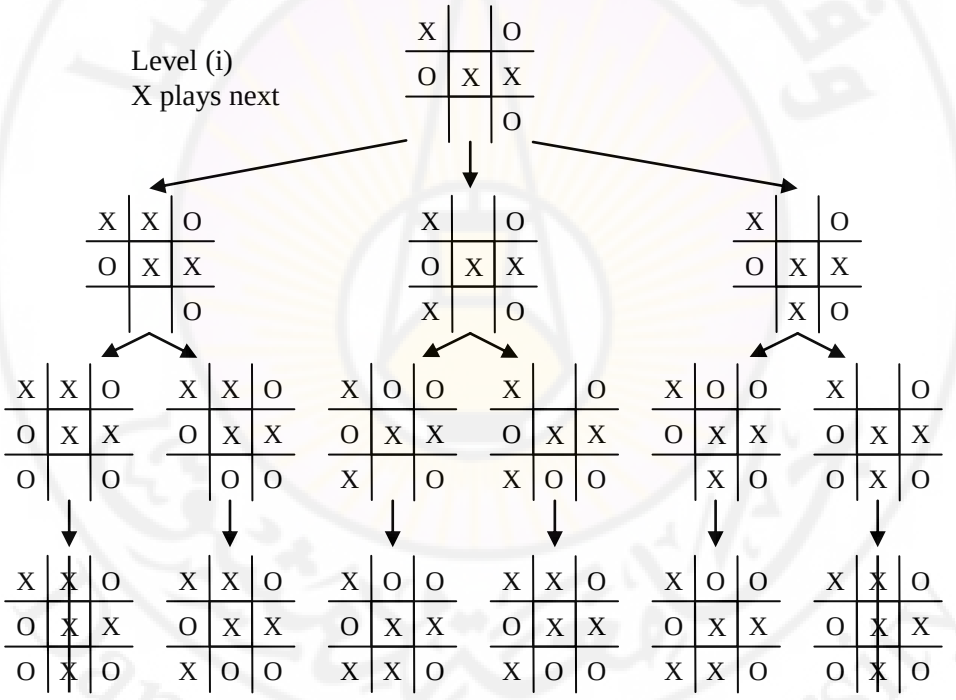
$$(u_b, u_s) = (100{,}450): Accepting$$
$$(u_b, u_s) = (0, 0): Rejecting$$

6. لعبة X–O (Tic–Tac–Toe):

لعبة بسيطة، يلعبها الأطفال بالورقة والقلم. وهي تلعب من قبل لاعبين، يدعى الأول (X) والثاني (O). يتم رسم شبكة من تسعة مربعات، ليقوم اللاعبان بشكل متتابع بملء المربعات بأسمائهم، ويكون الفائز هو الذي يسبق منافسه بتشكيل ثلاثة مربعات من اسمه مشكلة خطاً أفقياً أو شاقولياً أو قطرياً.

اكتشف لاعبو هذه اللعبة حديثاً، بأنه إذا لعب الخصمان بالطريقة الأفضل، سيؤدي ذلك إلى تعادل اللاعبين، ولذلك فإن (X-O) تلعب غالباً من قبل الأطفال، لينتقلوا بعد إتقانها لألعاب أكثر تعقيداً مثل (X-O with 12 Cell). يستخدم التمثيل الشجري في دراسة هذا النوع من الألعاب التتابعية التي تتضمن عدداً محدوداً من الخيارات أمام كل لاعب، وعليه اختيار الأفضل بناءً

على اختيار خصمه وعلى الاختيارات السابقة. يبين الشكل (5-3) تمثيلاً شجرياً للعبة في إحدى مراحلها المتقدمة، حيث يكون أمام اللاعب (X) ثلاثة خيارات، وعلي اختيار الحركة الأفضل. لا يفضل اختيار وضع (X) في الزاوية اليسرى من الأسفل، لأن ذلك سيؤدي بالنتيجة النهائية إلى التعادل. لكن يفضل وضع (X) في المنتصف من الأعلى أو الأسفل، لأن ذلك سيؤدي بالنتيجة النهائية إلى التعادل أو الفوز.



الشكل(5-3) التمثيل الشجري للعبة (X-O)

7. عروض شركات الاتصالات:

تقدم شركات الاتصالات عروضاً مغرية لزبائنها، ولنأخذ مثالاً عن تلك العروض في إحدى شركات الاتصالات المحمولة.

تقدم الشركة نوعين من عروض المكالمات للمشتركين هما:

- العرض الأول: أي مكالمة أقصر من (30 min) يكون سعرها (10 S.P)، وبعدها يتم فصل الخط.
- العرض الثاني: كل أسبوع يتم دفع (200 S.P) ويتم إعطاء المشترك (200 Min)، و(200 MB) من خدمة التواصل الاجتماعي (WhatsApp).

بالنظر إلى هذا العرض كلعبة، فإن المطلوب دراسة العرضين وإيجاد الأفضل بينهما.

سنقوم بإنشاء مصفوفة الحالات وفق الجدول المبين:

	عدم تفعيل العرض الثاني	تفعيل العرض الثاني
عدم تفعيل العرض الأول	ليس هذا المطلوب من الموضوع، لأن تكاليف الاتصالات عالية بدون العروض	دفع (200 S.P) أسبوعياً
تفعيل العرض الأول	عدم دفع شيء، لكن ينتهي العرض عند نهاية الشهر	حالة غير محققة، لأنه إذا تم تفعيل عرضين فسيتم تفعيل العرض الأخير، وإلغاء العرض السابق

قبل دراسة الحالات، نقوم بتطبيق (Nash Equilibrium) على اللعبة وإلغاء الحالات مستحيلة الاختيار، وهي حالة تفعيل كلا العرضين وحالة عدم تفعيلهما.

بدراسة كلا العرضين السابقين بشكل سريع نجد:

- إذا كان اللاعب من الأشخاص الذين يتحدثون بالهاتف بشكل مستمر، ولفترات زمنية طويلة نسبياً، يمكن القول عندها أن العرض الأول هو الأنسب له مبدئياً.
- العرض الثاني هو للأشخاص قليلي الكلام على الهاتف وكثيري استخدام برنامج (WhatsApp).

لنبدأ بدراسة متأنية لكلا العرضين السابقين.

- في العرض الأول يمكن للشخص تعبئة رصيد بقيمة اختيارية ولتكن (450 S.P). يمكن للشخص بوساطة هذا الرصيد أن يتكلم (45) مكالمة مدتها من (1 Sec) إلى (30 min)، وهنا يكون السؤال هل سيكون هذا الشخص من النوع الذي يتكلم على الجوال كثيراً لكن بفترات قصيرة أو بفترات طويلة أو حتى لا يتكلم نهائياً على جواله:

– إذا كان يتكلم لفترات قصيرة جداً ولنفترض (30 Sec) لكل مكالمة، عندها هذا العرض لن يكون بالجدوى التي يرغبها لأنه سيقوم بإعادة التعبئة ثلاث مرات شهرياً بتكلفة (1350 S.P)، وهي كلفة عالية مقارنة مع غيرها من التكاليف حيث قام الشخص في هذا الشهر بالكلام لمدة (67.5 min).

- إذا كان يتكلم لفترات طويلة نسبياً، ولتكن (10 min) وسطياً، عندها يمكن القول إن هذا العرض يناسب هذا النوع من المستخدمين حيث أن المستخدم يكون قد تحدث على جواله لمدة (1350 min) عند التعبئة ثلاث مرات شهرياً بتكلفة حوالي (1350 S.P)، والتي تعتبر قيمة جيدة نسبياً.
- إذا كان اللاعب من النوع الذي لا يتحدث بجواله عدداً كبيراً من المرات، فيكون هذا العرض أوفر له ومن الممكن أن يكلفه شهرياً من (225 S.P) إلى (500 S.P) وهي قيمة جيدة.

• في العرض الثاني يمكن للشخص في حال تعبئة (200 S.P) أسبوعياً أن يقوم بالحديث لمدة (200 min) في الأسبوع، أي ما يعادل (800 min) شهرياً بقيمة (900 S.P)، وبشكل مشابه يمكن استنتاج الحالات الآتية:

- إذا كان اللاعب من النوع الذي يتحدث كثيراً على جواله ولنفترض شهرياً لمدة (1350 min)، عندها هذا العرض سيكون مكلفاً للاعب أكثر من العرض السابق لأن في حال التعبئة الكاملة سيقوم بدفع (900 S.P) فقط، لكن إذا كان يريد تغطية زمن العرض السابق عندها سيقوم بدفع (1575 S.P) بالمجموع العام وهي قيمة أغلى من سابقتها، لكن ليس كثيراً بالأخص أن اللاعب يقوم بالحديث (50 min) زيادة عن العرض السابق.
- إذا كان اللاعب من النوع متوسطي الكلام على الجوال، عندها هذا العرض سيكون بتكلفة (900 S.P) شهرياً، ولن يتكلم سوى بقيمة (67.5 min) وهي ضمن الشريحة الأولى للأسبوع الأول

لكن إذا قام بتعبئة هذا الشهر فقط بقيمة (900 S.P) كما تم الذكر سابقاً سنجد أنها خيار أفضل للاعب، حيث أنها أوفر في المال له لأنه إذا قام بالتعبئة بقيمة (1500 S.P) فسيقوم باستهلاك الشهر الأول فقط (900 S.P) والشهر الثاني سيقوم بتعبئة فقط (300 S.P) بسبب الباقي من الشهر السابق.

- إذا كان اللاعب من النوع قليلي الكلام، عندها هذا العرض سيكون أنسب له حيث يقوم بتفعيله عند حاجته، لكن من الممكن أن يحتاجه مرة كل أسبوع عندها لن يكون هذا العرض مجدياً للاعب لأنه سيقوم بتكليفه أكثر من العرض السابق.

في النتيجة، نلاحظ بعد الدراسة كيف أن العرض الأول توقعنا أنه سيكون أفضل من ناحية الأشخاص الذين يتحدثون كثيراً على الجوال. وأن العرض الثاني أفضل للأشخاص قليلي الكلام على جوالهم. لكن بعد دراسة دقيقة للتكاليف قامت نظرية الألعاب بإثبات أن العرض الثاني أنسب للاعبين كثيري ومتوسطي الكلام، لكنه أسوأ للاعب الذي لا يتحدث سوى نادراً على جواله. وهنا نجد قوة نظرية الألعاب في مثال عملي بسيط.

8. لعبة حجرة ورقة مقص (Rock–Paper–Scissors):

وهي تندرج تحت الألعاب صفرية المجموع، أي أن الربح يتم عندما يكون المجموع صفر. إستراتيجية هذه اللعبة مختلطة (Mixed Strategies)، ومعناها أن يكون للاعب أهداف متعارضة ، ويقوم بالاختيار العشوائي بين الاستراتجيات المتاحة، ويقارن ربحه في كل مرة.

9. ابحث عني:

وهي لعبة تقوم على وضع فريقين في مدينة تمتلك عدداً محدداً من المعالم السياحية دون أن يتعرف الفريقان على بعضهما، ودون أن يعلموا بمكان وجودهم، تنتهي اللعبة عندما ينجح الفريق الذي يتوقع مكان الآخر، وينجح في كشفه قبل الثاني.

10. التهديد القابل للتصديق:

وهي لعبة تقوم على خلق هاجس الرعب لدى أحد اللاعبين، وذلك عندما يطلب اللاعب الأول طلباً من الثاني، مع وجود تهديد حقيقي قابل للتصديق ينفذ بحق الثاني إن لم ينفذ الطلب، وتبدأ اللعبة عندما تضع اللاعب الثاني في دوامة الخوف من إمكانية تنفيذك للتهديد.

في النهاية، بالنسبة للناحية الاقتصادية فتطبيقات نظرية اللعبة كبيرة ومتعددة للغاية كما أنها معقدة، فالأمور لا تكون أبداً بسيطة وسهلة كما هو الأمر في الأمثلة المطروحة أعلاه، لذا فهي تطبق عادة من قبل برمجيات مخصصة تتبع التغيرات وتتخذ القرارات تبعاً للتغيرات الحاصلة، ونادراً ما يتم تطبيقها من قبل البشر في الأنظمة شديدة التعقيد، أما التطبيقات السلوكية لها فهي قابلة للتطبيق بالاعتماد على تحليل شخصيات وظروف الأشخاص عموماً، لكن دقتها ليست مطلقة كون الأمر كثيراً ما يتضمن متغيرات عديدة تجعل حالات مثل معضلة السجن حالات خاصة وبسيطة جداً.

5-10- خوارزميات الألعاب:

بعد ظهور الحاسوب وتطوره وزيادة سرعته استخدم لبرمجة بعض الألعاب الشهيرة، حتى أضحى خصماً لا يستهان به، بعد تغلبه على أشهر اللاعبين، خاصة في الألعاب اللوحية الثنائية (Board Games) كالشطرنج و (X-O). يوجد العديد من الخوارزميات المستخدمة في برمجة ألعاب الحاسوب، كي تتمتع بالذكاء الذي يؤهلها لخوض المنافسة ضد البشر.

بعض الألعاب التتابعية يكون عدد حالاتها صغيراً نسبياً، بسبب محدودية الحركات والخيارات المتاحة، ما يعني إمكانية الحاسوب من فحص جميع الاستراتيجيات الممكنة والخيارات المتاحة لديه خلال زمن قصير جداً، خاصة في هذا العصر الذي وصلت فيه سرعة المعالجة إلى حدود خيالية. وهذا يعني عدم تخيل وجود لاعب بإمكانه اللعب أفضل من الحاسوب، لأن الحاسوب يختبر جميع الاحتمالات الممكنة لسير اللعبة.

المشكلة هي في الألعاب التتابعية التي يكون عدد حالاتها كبيراً جداً، بحيث لا يستطيع الحاسوب فحص جميع الخيارات الممكنة خلال زمن اللعب وقد يتطلب سنوات لاختيار الحركة اللاحقة، وهذا بالطبع غير عملي، فعند الحديث عن لعبة مثل لعبة (Go)، فإننا لن نفكر أبداً باستخدام خوارزمية مسح شامل لاختبار جميع الحالات الممكنة، نظراً إلى الزمن الطويل جداً الذي يحتاجه الحاسوب. فعلى سبيل المثال، فإن الزمن اللازم للبحث الكامل (كامل شجرة البيان) في لعبة الشطرنج على حاسوب عادي يستغرق أكثر من ألف قرن.

بناءً على ما سبق، لا بد من إيجاد خوارزميات ذكية، تستطيع اختيار الخطوة الأفضل دون الاضطرار إلى فحص جميع الخيارات الممكنة. فالبحث عن الحل بطريقة (Adversarial Search) يعتمد على إيجاد أفضل حركة في لعبة

مكونة من لاعبين. تستخدم هذه الطريقة في الألعاب التي يكون فيها تقدم اللاعب على الآخر قابلاً للقياس، ويكون فوز اللاعب هو خسارة للآخر ، وبذلك يكون المجموع صفرياً (Zero-Sum Game). تستخدم هذه الطريقة عادة مع الألعاب اللوحية كالشطرنج و(X-O)، ومن أشهر الخوارزميات المستخدمة في هذا المجال خوارزمية (MiniMax) وخوارزمية (Alpha-Beta Pruning)، التي تعد تحسيناً لخوارزمية (MiniMax).

1. خوارزمية (MiniMax):

1) مبدأ عمل الخوارزمية:

من أشهر خوارزميات برمجة الألعاب التي تم تطويرها وتحسينها للحصول على أداء عال وبزمن قياسي. عرفت هذه الخوارزمية بعدة تسميات مثل (MinMax) و(MM) و(Saddle Point). تستخدم خوارزمية (MinMax) في الذكاء الصنعي، وفي نظرية اتخاذ القرار، وفي مجال الإحصاء. قبل بدء شرح الخوارزمية، نستعرض بعض ا لمصطلحات العامة المستخدمة في خوارزميات البحث عموماً.

- الحالة الابتدائية (Initial State):

الوضع الابتدائي للعبة ، ففي لعبة الشطرنج يتم الانطلاق باللعبة بعد ترتيب القطع بشكل محدد وبدء اللعب من قبل اللاعب صاحب القطع البيضاء.

- تابع الانتقال (Successor Function):

التابع المسؤول عن توليد الحركات المتاحة (Legal Moves) في وضعية معينة (state) من اللعبة. يوجد قوانين تحدد الحركات المتاحة لكل قطعة والأماكن الممكن الانتقال إليها.

- اختبار العقد (Terminal Test):

عند الانتقال من حالة إلى أخرى، يجب اختبار العقدة أو الحالة التي وصلت إليها اللعبة لتحديد فيما إذا وصلت اللعبة إلى نهايتها بفوز أحد اللاعبين أو التعادل.

- تابع التقييم (Evaluation Function):

يستخدم تابع التقييم (التقويم) لإعطاء قيمة عددية لكل حالة يحدد من خلالها اللاعب المتقدم في اللعبة. يحسب تابع التقييم لعقد الأوراق ومن ثم تحدد الحركة المناسبة للاعب اعتماداً على قيمة تابع التقييم.

بفرض أن أحدهم يريد اللعب ضد الحاسوب المبرمج وفق خوارزمية (MiniMax)، فإنه يتم الاعتماد على المبدأين الآتيين:

- عندما يريد الحاسوب اللعب ، سيحاول اختيار حركة تزيد من تقييمه وفرصة فوزه.
- عندما يريد أن يلعب الخصم، فإنه سيحاول اختيار حركة تقلل من تقييم الحاسوب.

ومن هنا جاءت التسمية (MiniMax)، فالمقصود من (Mini) هو قيام الخصم بمحاولة التقليل من تقييم الحاسوب. والمقصود من (Max) هو محاولة الحاسوب زيادة تقييمه.

عندما يحين دور اللعب على الحاسوب فإنه يقوم بتجريب جميع الحركات المتاحة باستخدام تابع الانتقال للخطوة اللاحقة، وفي كل حركة تجرب الحركات المتاحة التي تليها، وهكذا حتى الوصول إلى عمق معين (Depth) محدد مسبقاً، وهو من العوامل التي يحدد من خلالها مستوى صعوبة اللعبة. بعد الوصول للعمق المحدد يتم تقييم حالة اللعبة باستخدام تابع التقييم، واختيار الحركة التي أدت إلى

حالة اللعبة ذات التقييم الأفضل، حيث تعد هي الحركة الأفضل. وعادة تكون قيم تابع التقييم موجبة للحالات التي تصب في مصلحة اللاعب (Max)، وسالبة للحالات التي تصب في مصلحة اللاعب (Min).

غالباً ما تستخدم مع خوارزمية (MiniMax) خوارزمية البحث بالعرض أولاً (Width First Search) أو خوارزمية بالعمق أولاً (Depth First Search). كما تحدد بعض الشروط للانتهاء مثل تجاوز الزمن المخصص للبحث، أو زيادة حجم البيانات عن الحجم المخصص في الذاكرة، أو الوصول إلى عمق محدد في شجرة البحث.

يستخدم التمثيل الشجري لتوضيح آلية عمل الخوارزمية بحيث تمثل العقدة الأولى في المستوي الصفري (Level 0) الجذر أو العقدة الابتدائية (Initial Node)، وهي تبين الحالة الابتدائية للعبة قبل بدء اللاعب الأول باختيار حركته. بينما تمثل بقية العقد الحالات المختلفة للعبة. كما تستخدم المسارات (Edges) لتوضيح الحركات الممكنة، والتي تنقل اللعبة من حالة إلى أخرى.

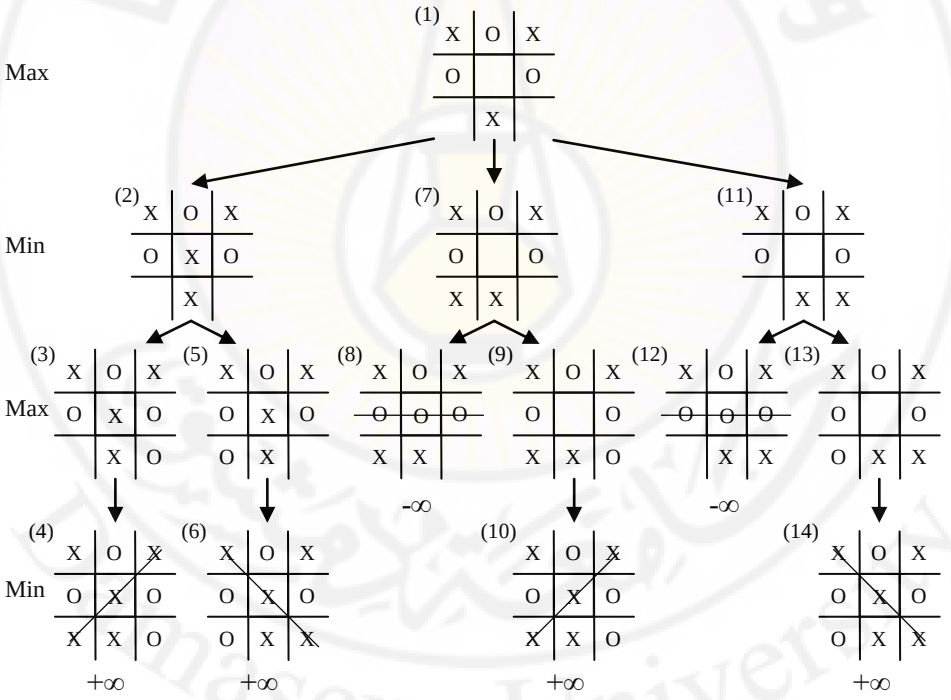
عندما يبدأ الحاسوب باللعب انطلاقاً من الحالة الابتدائية فإنه سيختار الحركة التي تعطيه أفضل تقييم، ولهذا فإن هذه العقدة تسمى (Max Node). نتيجة حركة الحاسوب، فإن اللعبة تنتقل إلى المستوي الأول (Level 1)، ليحين عندها دور الخصم في اللعب. يكون أمام الخصم مجموعة من الخيارات الممكنة، ومن دون أدنى شك، يجب عليه اختيار الحركة الأفضل بالنسبة له، والتي تعطي أسوأ تقييم بالنسبة للحاسوب، ولهذا فإن هذه العقدة تسمى (Min Node). وبذلك يتبادل الحاسوب والخصم أدوار اللعب حتى الوصول للعمق المطلوب، أو الوصول لنهاية اللعبة. تشكل العقد الطرفية (Terminal Nodes) أوراق الشجرة (Leaves)، وهي العقد النهائية التي تقيم حالة اللعبة عندها. وبالتالي يمكن القول

إن مستويات العمق الزوجية تخصص للاعب (Max)، ومستويات العمق الفردية تخصص للاعب (Min).

2) مثال تطبيقي على لعبة (X–O):

تستخدم خوارزمية (MiniMax) في برمجة لعبة (X–O)، والتي سنوضحها باعتبار الانطلاق من مرحلة متقدمة في اللعبة. أي أن العقدة الابتدائية سنختارها إحدى حالات اللعبة في المستوي (Level 6)، بهدف تبسيط الشرح.

ليكن الشكل(5–4) المعبر عن التمثيل الشجري للعبة في (Level 6).



الشكل(5–4) التمثيل الشجري للعبة (X–O) اعتباراً من (Level 6)

بفرض أن الحاسوب يمثل اللاعب (X)، وليكن دور اللعب على الحاسوب في (Level 6). يمثل (Root) الحالة الحالية للعبة والتي يريد الحاسوب اختيار الحركة التالية عندها. يوجد عدة خيارات ممكنة، ويجب اختيار الحركة التي تعطي الحاسوب أفضل تقييم، وعند كل مستوي تختبر جميع الخيارات الممكنة، مع الأخذ بالاعتبار أن الخصم يختار الحركة التي تعطيه أعلى تقييم، بينما تخفض من تقييم الحاسوب. يوضع تسمية على يسار كل مستوي هي (Max) أو (Min). يختار (Root) الحركة التي تعطي (Max)، بينما يختار الخصم في المستوي الأدنى الحركة التي تعطي (Min) بالنسبة للحاسوب. وهكذا تتكرر الأدوار حتى الوصول إلى العمق (Depth) المطلوب أو انتهاء اللعبة.

عند الوصول إلى العقدة الطرفية رقم (4) مثلاً يتم تقييم اللعبة، وبما أنها فوز للحاسوب فإنها تقيم (∞+)، ولأن العقدة رقم (3) هي (Max) فإن الحاسوب سيختار القيمة الأعلى (والوحيدة) القادمة من العقدة رقم (4).

أما عند الوصول إلى العقدة رقم (2) مثلاً، فيلاحظ أنها (Min)، لذلك تختار القيمة الأقل بين (3) و(5)، ولكن لا فرق بينهما لأن كليهما سيؤديان إلى تقييم (∞+) والتي تعبر عن فوز الحاسوب، أي أنه لا مفر للخصم من الخسارة.

أما بالنسبة للعقدة رقم (7) فهي (Min)، لذلك تختار القيمة الأقل بين (8) و(9)، وهي (8) بقيمة (∞−)، والتي تعبر عن خسارة الحاسوب وفوز الخصم.

بعد سلسلة هذا النوع من الاختبارات يستنتج الحاسوب أن أفضل حركة هي الانتقال من الجذر إلى العقدة رقم (2).

إن لعبة (X–O) بسيطة ولا تحتاج إلى طرائق كثيرة ومعقدة لتقييم حالة اللعبة، كلعبة الشطرنج المعقدة التي تستخدم كثيراً من الخصائص (Features) لتقييم وضعية اللعبة، فمثلاً حاسوب (Deep Blue) من (IBM) تغلب في عام

(1996) على بطل العالم للشطرنج (Garry Kasparov)، باستخدام 8000 (Features) لتقييم اللعبة ، مثل عدد القطع المتبقية في اللعبة، و أنواع القطع، وأماكن القطع، ومحيط أمان الملك، وتركيب البيادق (Pawns Structure)، والتحكم بالوسط، وغيرها الكثير.

في لعبة (X–O) يكون تابع التقييم (Evaluation Function) بسيطاً، ويركز على عدد الفرص المتاحة للفوز ومدى القرب منها. فبفرض أن الحاسوب يلعب دور (X)، يمكن وضع التقييمات الآتية:

- لكل مربعين في خط واحد محجوزين من قبل (X) والمربع الثالث فارغ نضع (+10).
- لكل مربع في خط واحد محجوز من قبل (X) والمربعين الآخرين فارغين نضع (+1).
- لكل مربعين في خط واحد محجوزين من قبل (O) والمربع الثالث فارغ نضع (−10).
- لكل مربع في خط واحد محجوز من قبل (O) والمربعين الآخرين فارغين نضع (−1).

كما يمكن تجريب تقييمات أخرى كأن نقول، ليكن تابع التقييم (F(p)) للموقع (p) على لوحة (X–O) كما يأتي:

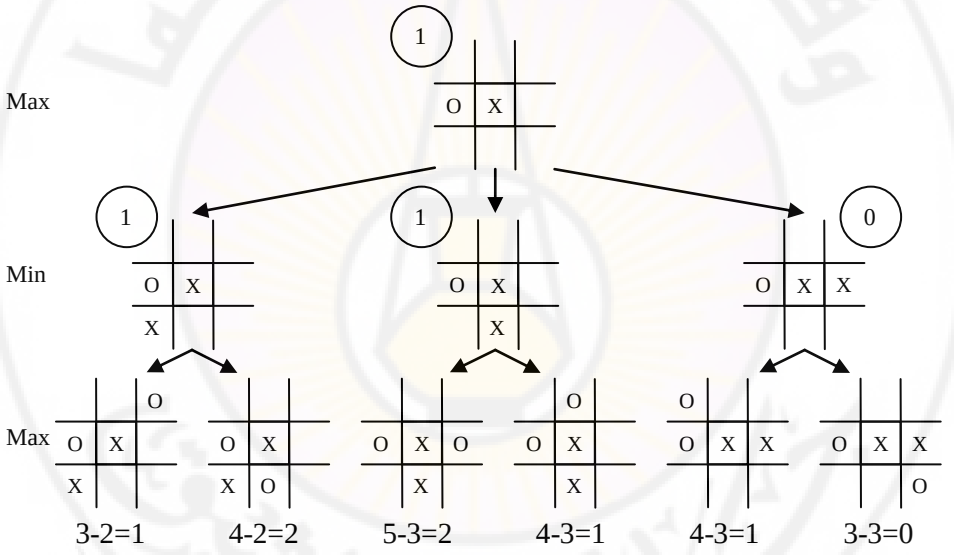
إذا كان (p) موقعاً رابحاً للاعب (Max)، فإن (F(p) = +∞).

إذا كان (p) موقعاً رابحاً للاعب (Min)، فإن (F(p) = −∞).

إذا كان (p) موقعاً غير رابح لأي من اللاعبين فإن (F(p) = A–B)، حيث أن (A) هي عدد الصفوف والأعمدة والأقطار الكاملة التي بقيت مفتوحة

للاعب (Max). أما (B) فهي عدد الصفوف والأعمدة والأقطار الكاملة التي بقيت مفتوحة للاعب (Min).

يبين الشكل(5-5) جزءاً من شجرة لعبة (X-O)، بهدف توضيح كيفية حساب تابع التقييم. تكتب قيمة تابع التقييم للعقد في أسفل كل عقدة طرفية في الشكل المبين فقط. كما تكتب القيمة المختارة ضمن دائرة، حيث يتم اختيار القيمة الأدنى في مستوي (Min)، بينما يتم اختيار القيمة الأعلى في مستوي (Max)،



الشكل(5-5) جزء من التمثيل الشجري للعبة (X-O) مع تقييم العقد

تعد الحركات المتاحة في لعبة (X-O) قليلة وتسمى في التمثيل الشجري (Branching Factor)، لذلك يمكن وضع العمق (Depth = 9) لضمان الوصول إلى نهاية اللعبة دائماً . أما في الألعاب المعقدة فيكون عدد الحركات كبيراً جداً كالشطرنج، ما يتطلب من الحاسوب زمناً طويلاً للتفكير في

جميع الخيارات الممكنة وتفرعاتها، و لذلك أجريت بعض التحسينات على الخوارزمية لتجنب أخذ العقد غير الضرورية بالحسبان. وقد أطلق على الخوارزمية الجديدة اسم (Alpha-Beta Pruning).

2. خوارزمية (Alpha-Beta Pruning):

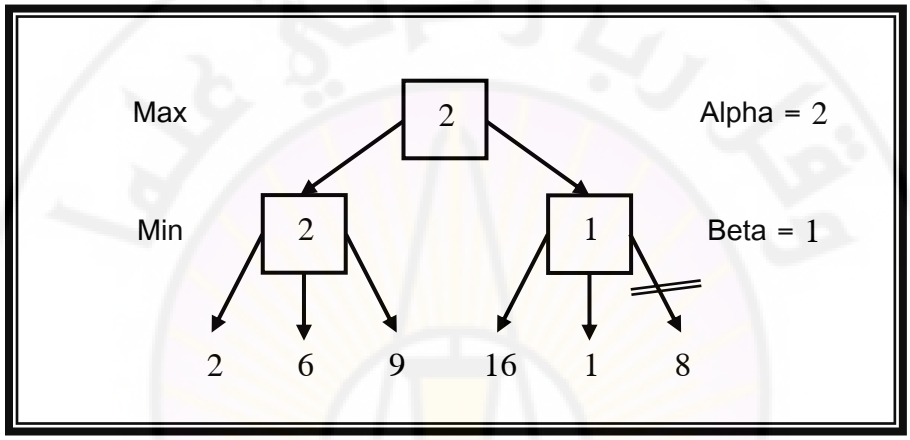
1) مبدأ عمل الخوارزمية:

تصنف كخوارزمية بحث متطورة عن خوارزمية (MiniMax)، حيث تحاول التخفيض من عدد العقد المنشأة بوساطة خوارزمية (MiniMax)، وهي تستخدم في الألعاب ثنائية اللاعبين كألعاب (Chess, Go, X-O)، والتي يحاول فيها كل لاعب اختيار الأفضل له والأسوأ للخصم. تقوم فكرتها على إيقاف تطور عقد الشجرة عندما تجد أن الحركة اللاحقة سوف تكون أسوأ مما سبقها، وبالتالي تجري عملية تقليم للأفرع غير المؤثرة على النتيجة النهائية. تؤدي عملية التقليم إلى تخفيض حجم الذاكرة اللازم لتخزين عقد الشجرة، وإلى تخفيض زمن المعالجة، وبالتالي زيادة كفاءة وأداء عملية البحث عن الخيار الأفضل.

تستخدم خوارزمية البحث بالعمق أولاً (Depth First Algorithm) كإستراتيجية بحث في خوارزمية (Alpha-Beta)، أما عند استخدام خوارزمية البحث الأفضل أولاً (Best First Algorithm) كإستراتيجية بحث، فإننا نحصل على خوارزمية (SSS) التي تنفذ بزمن أقصر عموماً، مقابل استهلاك حجم ذاكرة أعلى.

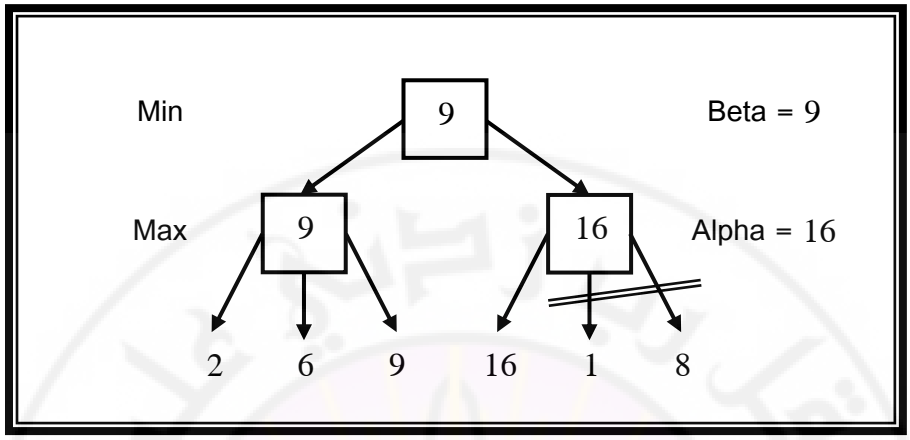
يعطى اللاعب (Max) المتغير (Alpha)، واللاعب (Min) المتغير (Beta)، أو بشكل رمزي (α, β). كما يشترط للتوقف عن متابعة عملية البحث تحقق إحدى الشرطين الآتيين:

- عندما تكون قيمة (β) أقل أو تساوي قيمة (α) في المستوى الأعلى للاعب (Min)، فإننا لا نقوم بإنشاء أفرع للشجرة أسفل اللاعب (Min)، وهو ما يسمى بقطع بيتا (Beta Cutoff)، المبين بالشكل(5-6).



الشكل(5-6) قطع بيتا (Beta Cutoff)

- عندما تكون (α) أكبر أو تساوي قيمة (β) في المستوى الأعلى للاعب (Max)، فإننا لا نقوم بإنشاء أفرع للشجرة أسفل اللاعب (Max)، وهو ما يسمى بقطع ألفا (Alpha Cutoff)، المبين بالشكل(5-7).



الشكل(5-7) قطع ألفا (Alpha Cutoff)

تشير قيمة (Alpha) إلى أدنى قيمة مضمونة للاعب (Max)، بينما تشير قيمة (Beta) إلى أعلى قيمة مضمونة للاعب (Min).

تعطى (Alpha) القيمة الابتدائية (∞-) فهي حتماً قيمة مضمونة للاعب (Max)، لأنه لا يوجد أقل منها. بينما تعطى (Beta) القيمة الابتدائية (∞+)، وهي أيضاً قيمة مضمونة للاعب (Min)، لأنه لا يوجد أكبر منها، وهذا ما يضمن انطلاق اللاعبين من أسوأ قيمة ممكنة.

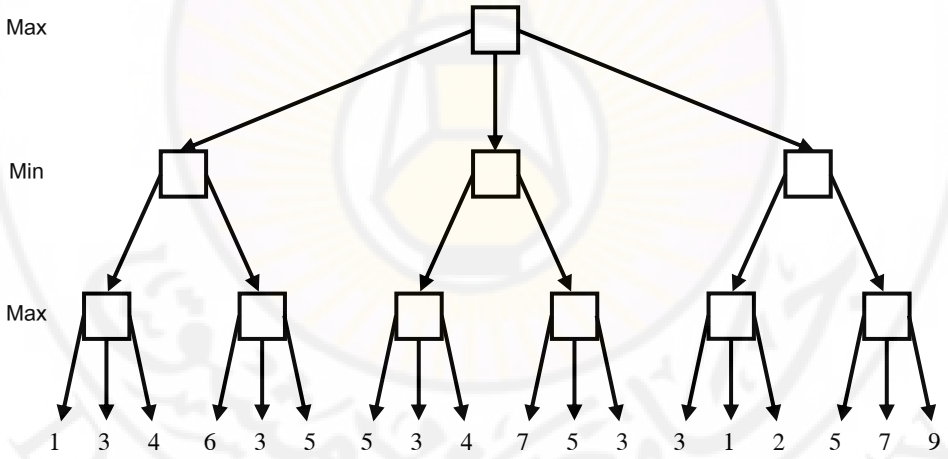
كلما ضمنا أن تصبح القيمة العليا للاعب (Min)، أي (Beta)، أقل من القيمة الدنيا للاعب (Max)، أي (Alpha)، فإن اللاعب (Max) لن يحتاج التفرع من عقدته الحالية. عملياً، فإن العملية تشبه حالة لاعب الشطرنج الذي حان دوره في اللعب، ووجد أن الحركة الأولى ولتكن (A) تحسن موقعه في اللعبة. وبمتابعة البحث عن بقية الاحتمالات، وجد أن حركة ثانية (B) جيدة أيضاً، لكنها تؤدي إلى موت الملك وخسارته اللعبة من قبل الخصم في حركتين تاليتين. بعد هذا الاستنتاج لن يفكر اللاعب الأول باختيار الحركة (B) والمغامرة بخسارة اللعبة.

نوضح فيما يأتي ترميزاً برمجياً لخوارزمية (Alpha–Beta)، مع استخدام خوارزمية البحث بالعمق أولاً كإستراتيجية بحث.

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut–off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if α ≥ β then
                break (* α cut–off *)
        return value
(* Initial call *)
alphabeta(origin, depth, −∞, +∞, TRUE)
```

2) مثال على خوارزمية (Alpha–Beta):

يوضح المثال الحالي كيفية استخدام خوارزمية (Alpha–Beta) بهدف تقليل عدد العقد المختبرة. لتكن الشجرة المبينة في الشكل(5–8)، حيث كتب على العقد الطرفية قيمة تابع التقييم. لتبسيط عملية الشرح فإننا نرقم مستويات العقد ابتداءً من الجذر (Level 0)، وباتجاه الأسفل حتى الوصول إلى (Level 2). كما أن العقد ترقم في كل مستوٍ ابتداءً من العقدة الأولى في أقصى اليسار وحتى العقدة اليمنى.



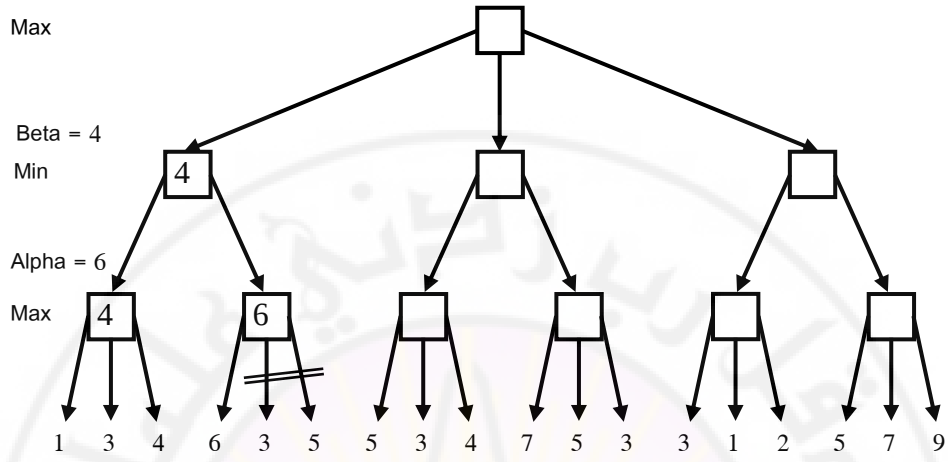
الشكل(5–8) شجرة مثال خوارزمية (Alpha–Beta)

ليكن المستوي (Level 0) مخصص للاعب (MAX)، وباعتبار أن إستراتيجية البحث هي خوارزمية البحث بالعمق أولاً، فإن مسار البحث يبدأ من (Level 0) إلى العقدة الأولى من (Level 1)، فالعقدة الأولى من (Level 2).

الآن تبدأ الخوارزمية بفحص العقد الطرفية ابتداءً من الورقة الأولى بقيمة تابع تقييم يساوي (1) وهي قيمة أكبر من (Alpha = −∞)، لذلك تصبح (Alpha = 1). وبما أن (Beta ≤ Alpha) غير محققة، فإن الخوارزمية تتابع إلى الورقة الثانية بقيمة تابع تقييم يساوي (3)، وهي أكبر من (Alpha = 1)، لذلك تصبح (Alpha = 3)، وما زالت (Beta ≤ Alpha) غير محققة. تتابع الخوارزمية إلى الورقة الثالثة بقيمة تابع تقييم يساوي (4)، وهي أكبر من (Alpha = 3)، لذلك تصبح (Alpha = 4)، وما زالت (Beta ≤ Alpha) غير محققة.

تتابع الخوارزمية بالرجوع إلى (Level 1) وهو مستوي (MIN). الآن، بما أن القيمة الابتدائية (Beta = +∞) وقيمة (Alpha = 4)، فإنه يتم اختيار القيمة الأصغر أي (Beta = 4).

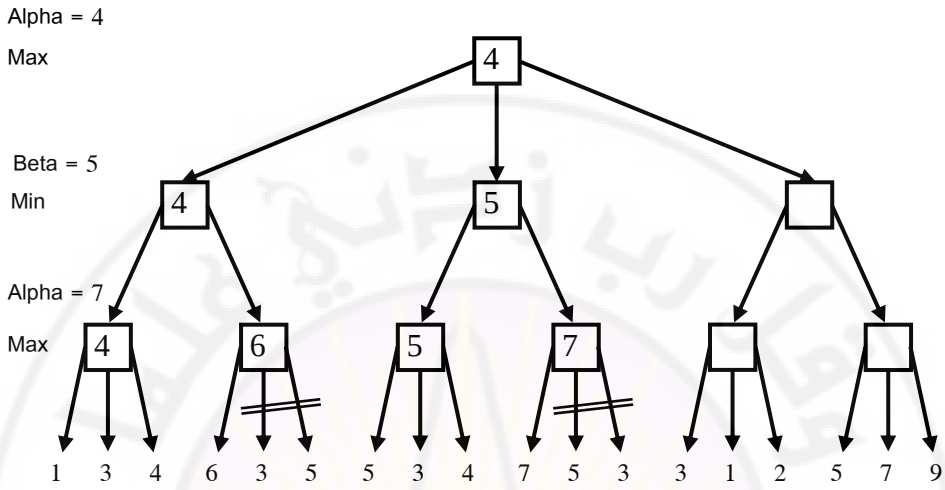
تعاود الخوارزمية النزول إلى (Level 2) وهو مستوي (MAX)، لاختبار العقدة الثانية في (Level 2). يتم البدء من الورقة الأولى ذات تابع التقييم (6). وبما أن القيمة الابتدائية (Alpha = 4) وقيمة تابع التقييم أكبر من قيمة (Alpha = 4)، فإنه يتم اختيار القيمة الأكبر أي تصبح (Alpha = 6). باختبار الشرط (Beta ≤ Alpha) نجده محققاً (6 ≤ 4)، لذلك يتم تنفيذ (Alpha Cutoff)، وعدم اختبار ما تبقى من الأوراق في العقدة الثانية من (Level 2). هذا منطقي، فالمستوي (MAX)، ومهما كانت قيم تابع التقييم لبقية الأوراق فلن نختارها إذا كانت أقل من (6). وإن كانت أكبر (6)، فلن يتم اختيارها في المستوي الأعلى لأنه يختار (MIN) وهي حالياً (4)، وهذا ما يبينه الشكل(5-9).



الشكل(5-9) الخطوة (a) في شجرة مثال خوارزمية (Alpha-Beta)

تتابع الخوارزمية بالرجوع إلى (Level 1)، وبما أنه لا يحتوي على عقدة أخرى فرعية فإنها تتابع إلى (Level 0). بما أن (Level 0) مستوي (MAX)، فإن (Alpha = 4) في (Level 0).

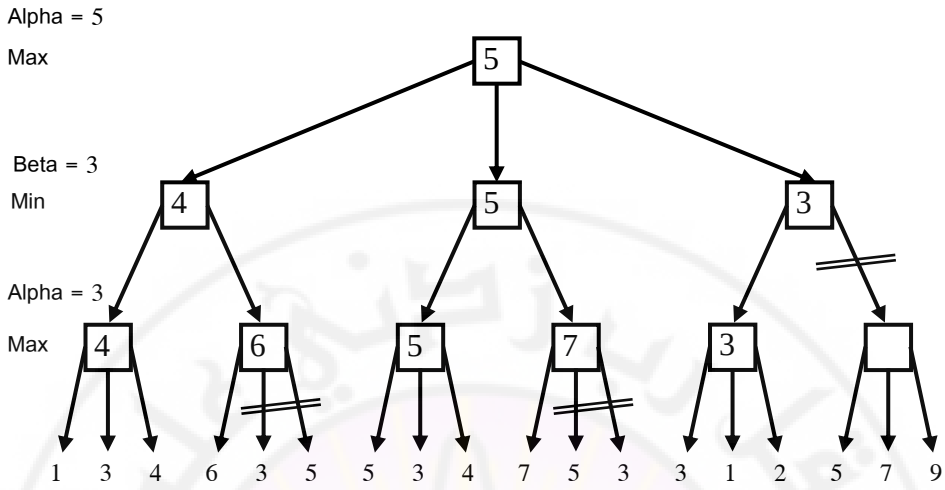
تعاود الخوارزمية النزول إلى (Level 1) ثم إلى (Level 2) وهو مستوي (MAX). بالطريقة نفسها، فبعد اختبار أوراق العقدة الثالثة تصبح (Alpha = 5) في المستوي (Level 2). بالعودة إلى (Level 1) تصبح (Beta = 5)، ومن ثم تنزل الخوارزمية إلى العقدة الرابعة في (Level 2)، حيث تصبح (Alpha = 7). باختبار الشرط (Alpha $\leq$ Beta) نجده محققاً ($7 \leq 5$)، لذلك يتم تنفيذ (Alpha Cutoff)، وعدم اختبار ما تبقى من الأوراق في العقدة الرابعة من (Level 2)، وهذا ما يبينه الشكل(5-10).



الشكل(5-10) الخطوة (b) في شجرة مثال خوارزمية (Alpha-Beta)

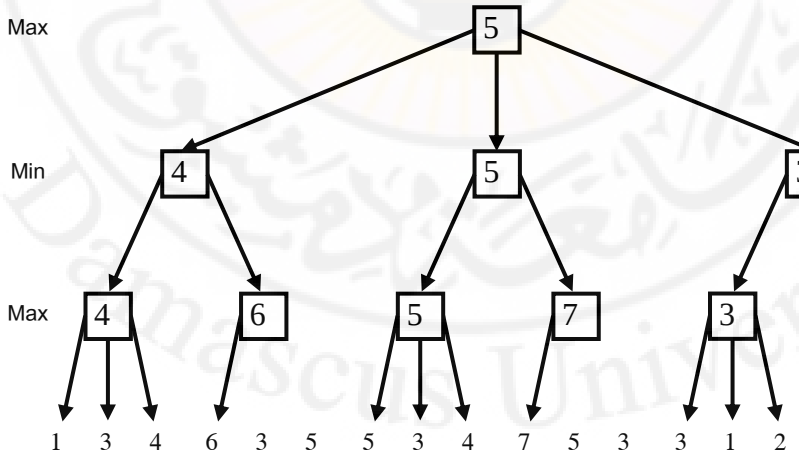
تتابع الخوارزمية بالرجوع إلى (Level 1)، وبما أنه لا يحتوي على عقدة أخرى فرعية فإنها تتابع إلى (Level 0). بما أن (Level 0) مستوي (MAX)، فإن (Alpha = 5) في (Level 0).

تعاود الخوارزمية النزول إلى (Level 1) ثم إلى (Level 2) وهو مستوي (MAX). بالطريقة نفسها، فبعد اختبار أوراق العقدة الخامسة تصبح (Alpha = 3) في المستوي (Level 2). بالعودة إلى (Level 1) تصبح (Beta = 3)، وقبل نزول الخوارزمية إلى العقدة السادسة في (Level 2)، فإنه يختبر الشرط (Beta ≤ Alpha)، فنجده محققاً ($3 \leq 5$)، لذلك يتم تنفيذ (Beta Cutoff)، وعدم اختبار ما تبقى من العقد في (Level 2)، وهذا ما يبينه الشكل(5-11).



الشكل(5-11) الخطوة (c) في شجرة مثال خوارزمية (Alpha-Beta)

لم يبق عقد في (Level 1)، وبالتالي تنتهي الخوارزمية، لنحصل على شجرة بعدد عقد أقل، وبالتالي بحجم ذاكرة أقل مما تتطلبه خوارزمية (MiniMax)، وهذا ما يبينه الشكل(5-12).



الشكل(5-12) الشجرة النهائية في مثال خوارزمية (Alpha-Beta)

وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل السادس

6- الخوارزميات الجينية (Genetic Algorithms):

6-1- مقدمة:

تعد الخوارزميات الجينية (الوراثية) (Genetic Algorithms) (أو اختصاراً GAs)، من التقنيات الهامة في البحث العشوائي عن الخيار الأمثل من مجموعة حلول متوافرة لتصميم معين. كما أنها أحد أقسام التطور في علوم الحاسوب التي بدأت تنتشر بسرعة كجزء من الذكاء الصنعي.

إن الخوارزميات الجينية، ما هي سوى محاكاة لعملية التزاوج بين الكائنات الحية من النوع نفسه، وهي تحاول بهذه الطريقة الوصول إلى الحل الأنسب للمشكلة المطروحة، بالحفاظ على الصفات (المزايا) الجيدة الموجودة في جيل الآباء لنقلها إلى جيل الأبناء، بهدف الحصول على ذرية قوية تتمتع بأفضل صفات جيل السلف على أقل تقدير.

تعد نظرية داروين في التطور والاصطفاء الأساس في تطور الخوارزميات الجينية، إذ تقول ببساطة إن المشاكل التي تعاني منها الكائنات الحية، يمكن إيجاد حل لها عن طريق التطور والانتقاء الطبيعي، بمعنى آخر "البقاء للأنسب". من هنا يصبح مبدأ الخوارزميات الجينية "البقاء للحل الأمثل". وبذلك فهي تمثيل للاعتقاد السائد بأن الذكاء البشري يخلق مع الإنسان ويتم اكتسابه عن طريق الوراثة بشكل كبير.

تقوم المعالجة الوراثية بتمرير الصفات القوية من خلال عمليات التوالد المتعاقبة وتدعيم هذه الصفات. حيث يكون لهذه الصفات القدرة الأكبر على دخول عملية التوالد وإنتاج ذرية قوية، وبتكرار الدورة الوراثية تتحسن نوعية الذرية تدريجياً وتتجه نحو إنتاج جيل مثالي يتمتع بأفضل الصفات.

الخوارزميات الجينية (GA) هي طريقة لمحاكاة ما تفعله الطبيعة في تكاثر الكائنات الحية، واستخدام تلك الطريقة لحل مشكلات معقدة للوصول إلى الحل الأفضل أو أقرب حل ممكن له. تستخدم (GA) لحل المشاكل المعقدة، وذلك من أجل تعليم الآلة وكذلك من أجل تطوير البرامج، كما تستخدم في بعض أنواع الفنون كالرسم وتأليف المقطوعات الموسيقية وفي معالجة الإشارات أحادية البعد ومتعددة الأبعاد كالصور . تتبع فائد ة (GA) في مطابقتها للواقع، وهي سهلة التعامل ويكفي أن تتعلم قواعدها التي تبدأ من صياغة صبغيات حتى تحل مشاكل أخرى. إلا أن سيئة (GA) في زمن التنفيذ، حيث تعد أبطأ من بلقي طرائق الذكاء الصنعي، لكن بالمقابل يمكن إيقاف التنفيذ بأي وقت كان، كما أن إطالة الزمن مقبول حالياً، بسبب التقدم العلمي وازدياد سرعة المعالجات بشكر كبير كل فترة وجيزة نسبياً، كما أن الزمن الطويل على الحواسيب الشخصية لم يعد ذو أهمية، نظراً لتوافر الحواسيب الخاصة مثل (Super Computer, Main Frame).

لتطبيق الخوارزمية الجينية يجب أولاً إيجاد التمثيل المناسب للمشكلة المدروسة وفق عمليات رياضية لتمثيل الصبغيات (Chromosomes)، حيث تعد طرائق التمثيل باستخدام السلاسل الثنائية من أشهرها. بعد تمثيل قيم المتغيرات المعبرة عن حل المشكلة المعطاة على هيئة صبغيات ، وبعد أن تنتج هذه الصبغيات الممثلة للحلول بواحدة من طرائق الترميز، لا بد من طرائق لمعالجتها بوساطة عدة عمليات رياضية مستنبطة من العمليات البيولوجية كالانتخاب والعبور والطفرة، للحصول في نهاية المطاف على مجموعة من الصبغيات التي تمثل الجيل النهائي، وكل صبغي ما هو إلا فرد من أفراد الجيل، وأفضل صبغي هو الحل الأمثل، الذي نبحث عنه للمسألة المطروحة.

إذن، تبدأ عملية البحث انطلاقاً من مجموعة حلول ، وليس من حل أو نقطة واحدة. فالخوارزمية الجينية مبنية على أساس تقنية الحلول المثلى التي تحاكي النشوء الطبيعي عن طريق تشفير الحلول الممكنة لتمثيلها على شكل سلاسل مشابهة لسلاسل الصبغي، ومن ثم تطبيق بعض العمليات البيولوجية (نسخ، طفرة) لإنتاج الحل الأمثل. تعد الطبيعة التكيفية في الخوارزمية الجينية الميزة الأهم التي تجعلها أقل حاجة لمعرفة المعادلة الممثلة للمسألة من أجل حلها. فالخوارزميات الجينية هي طريقة لمحاكاة ما تفعله الطبيعة في تكاثر الكائنات الحية واستخدام تلك الطريقة لحل مشكلات معقدة للوصول للحل الأفضل، أو أقرب حل ممكن للحل الأفضل. وقد استعارت عدة مصطلحات وصفات من علم الوراثة كالجيل والوالدين والعبور والطفرة. ففكرة الخوارزميات الجينية تكمن في توليد بعض الحلول للمشكلة عشوائياً، ثم فحص هذه الحلول ومقارنتها ببعض المعايير التي يضعها مصمم الخوارزمية، وأفضل الحلول فقط هي التي تبقى، أما الحلول الأقل كفاءة فيتم إهمالها عملاً بالقاعدة البيولوجية "البقاء للأصلح". والخطوة التالية هي مزاوجة أو خلط الحلول المتبقية (الحلول الأكثر كفاءة) لإنتاج حلول جديدة على غرار ما يحصل في الكائنات الحية، وذلك بمزج مورثاتها (جيناتها) بحيث يحمل الكائن الجديد صفات عبارة عن مزيج من صفات والديه. الحلول الناتجة من التزاوج تدخل أيضاً تحت الفحص والتنقيح لمعرفة مدى كفاءتها واقترابها من الحل الأمثل، فإن ثبتت كفاءة الحل الجديد فإنه يبقى وإلا يتم إهماله، وهكذا تتم عمليات التزاوج والانتقاء حتى تصل العملية إما لعدد معين من التكرارات (يقرره مستخدم النظام) أو تصل الحلول الناتجة أو إحداها إلى نسبة كفاءة أو نسبة خطأ ضئيلة (يحددها المستخدم أيضاً).

6-2- تقنيات البحث:

لقد أثبتت الخوارزميات الجينية قدرتها على حل العديد من المشاكل، وتغلبت في قدراتها على طرائق البحث التقليدية، مثل خوارزمية تسلق الهضبة (Hill Climbing). بالطبع هناك عدة تقنيات للبحث عن حل للمسألة المطروحة. يبين الشكل(6-1) موقع الخوارزميات الجينية في شجرة تقنيات البحث، حيث تقسم تقنيات البحث إلى ثلاث تقنيات أساسية؛ هي:

7. تقنيات البحث الرياضية (Calculus-based Techniques):

يوجد طريقتا بحث رياضية؛ هما:

3) طريقة البحث الرياضية المباشرة (Direct Methods):

يتم الوصول للحل بشكل متسلسل بما يشبه عملية تسلق الجبال خطوة فخطوة نحو القمة. تستخدم في هذه الطريقة بعض المتسلسلات الرياضية وبالتالي الاقتراب من الحل يكون بشكل تدرجي . يوجد عدة خوارزميات لطريقة البحث الرياضية المباشرة وهي:

1. طريقة فيبوناتشي (Fibonacci):

2. طريقة نيوتن (Newton):

4) طريقة البحث الرياضية غير المباشرة (Indirect Methods):

وهي تتم بمساواة مجموعة من المعادلات للصفر ، ثم إجراء بعض العمليات الرياضية البسيطة للحصول في النهاية وبعد عدة دورات على الحل.

8. تقنيات البحث العشوائي الموجه (Guided random search Techniques):

عادة يكون فضاء الحل واسعاً، ما يعني صعوبة البحث عن حل أمثل للمسألة من عدد كبير من الحلول الممكنة. تعتمد هذه التقنية على أخذ عينات عشوائية من الحلول الممكنة واستخدامها في الاقتراب والوصول للحل الأمثل دون الاضطرار لتجريب كل الخيارات الممكنة.

يوجد طريقتا بحث عشوائي؛ هما:

1) خوارزميات التطور (Evolutionary Algorithms):

تستخدم خوارزميات التطور في علوم الحاسوب لإيجاد الحل الأمثل، حيث تقسم إلى قسمين هما:

1. استراتيجيات التطور (Evolutionary Strategies):
2. الخوارزميات الجينية (Genetic algorithms):

بدورها تقسم الخوارزميات الجينية إلى:

1. تفرعية (Parallel):

إما أن تكون مركزية (Centralized)، أو أن تكون موزعة (Distributed).

2. تسلسلية (Sequential):

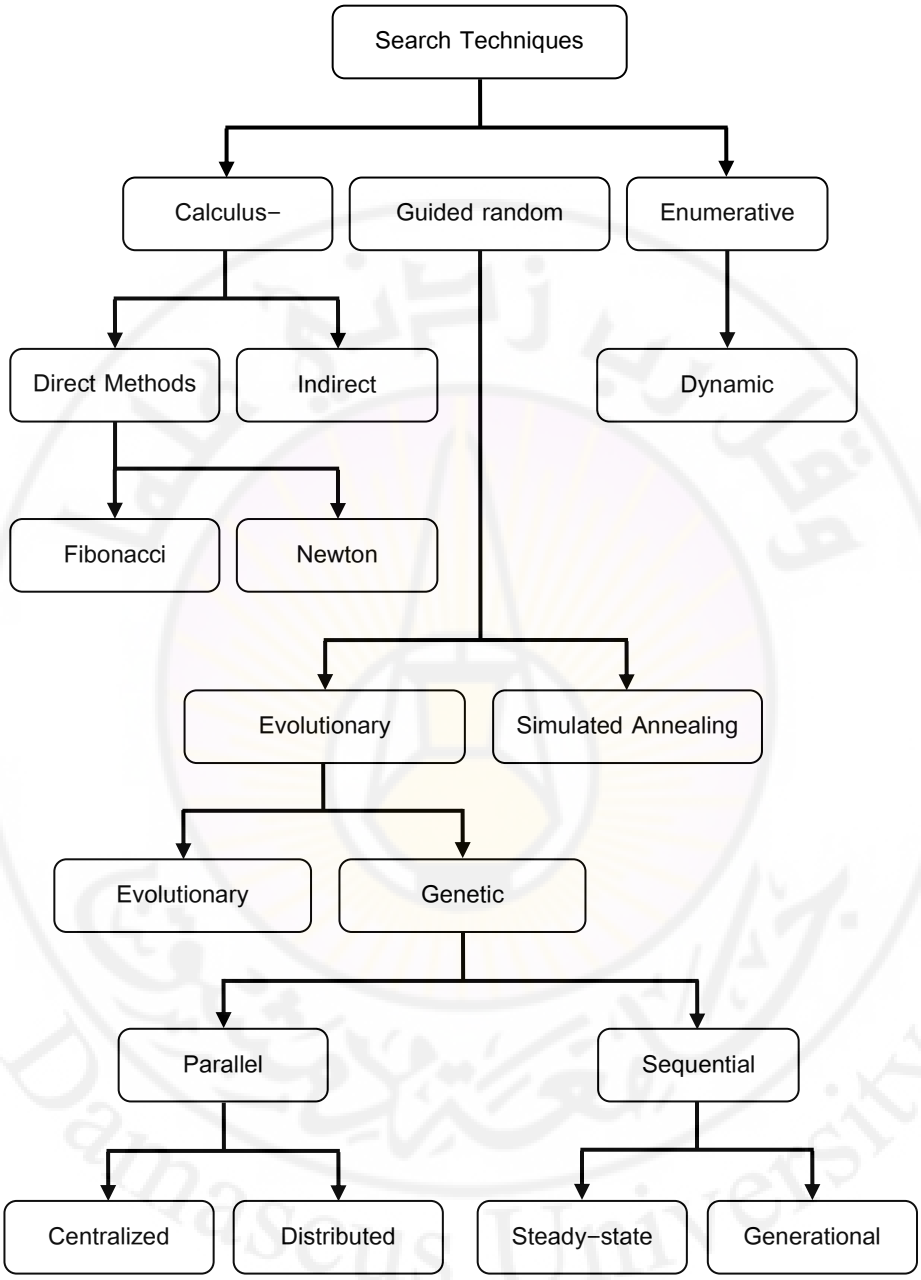
إما أن تكون بحالة ثابتة (Steady-state)، أو أن تكون في حالة توالد ونشوء (Generational).

2) خوارزميات المحاكاة (Simulated Annealing Algorithms):

9. تقنيات التعداد الكامل (Enumerative Techniques):

المبدأ العام لهذه الطريقة هو إيجاد الحل الأفضل لمشكلة معينة، وذلك بوساطة مسح (Scan) جميع الخيارات الممكنة ومن ثم مقارنة الحلول وإيجاد الأفضل منها، وفي هذه الحالة يجب أن يكون المجتمع محدوداً.

والمشكلة تظهر كلما زاد عدد المتغيرات لحل المسألة، وفي بعض المسائل قد يستحيل مسح جميع الحلول بسبب التكلفة المادية أو الزمنية.



الشكل(6–1) شجرة تقنيات البحث

6-3- بدايات الخوارزميات الجينية:

ركزت التجارب في الذكاء الصنعي على محاولة تكرار تصرفات الإنسان (أذكى الكائنات الحية) وتطبيقها في مجال البرمجيات، وقد استطاعت هذه المقاربة نوعاً ما أن تحقق نجاحاً ملحوظاً، وأكبر مثال على ذلك آلة لعب الشطرنج المسماة (Deep Blue Chess Machine) التي تغلبت على الذكاء البشري المتمثل باللاعب (Kasparov) وذلك في شهر أيار من عام 1997. لكن عملية المحاكاة السابقة للسلوك البشري كانت محدودة نوعاً ما، حيث وقفت عاجزة عن حل بعض المسائل التي يعرف معظم الناس حلها مسبقاً. ومن هنا بدأت تظهر فكرة الطرائق الذكية الحسابية (Computational Intelligence Methods) مثل الحوسبة التطورية (Evolutionary Computing)، ومع بدء تطور علوم الحاسوب في ستينيات القرن الماضي بوساطة العديد من مراكز الأبحاث والعلماء مثل (I. Rechenberg)، الذي بدأ بالتحدث عن استراتيجيات التطور ، ومن ثم طورت فكرته من قبل عدة باحثين آخرين. لقد تمكن هذا العلم من حل المسائل المعقدة دون الاعتماد على خبرة الإنسان، وإنما تم الاستفادة من آلية التطور المطروحة في نظرية داروين وتحويلها لنموذج حاسوبي كإجرائية للأمثلة، فكما في الطبيعة، فإن عملية التطور في الكائنات الحية تهدف للتكيف مع البيئة المحيطة بهدف النجاة. فعملية التطور تتجه دوماً نحو ما هو أمثل وأفضل للكائن الحي، ومثال عليها تطور الزرافات بحيث استطالت أعناقها لتستطيع الوصول إلى غذائها المتمثل بأوراق الأشجار العالية، إذ إن البقاء للأصلح.

وفعلاً، لم تلبث الأفكار السابقة طويلاً حبيسة المختبرات، حيث تم فعلياً طرح فكرة الخوارزميات الجينية لأول مرة ، والتي هي جزء من الحوسبة التطورية ، بشكل رسمي في الولايات المتحدة الأمريكية من قبل بروفيسور في علوم الحاسوب

من جامعة ميشيغان (University of Michigan) يدعى جون هولاند (Johon Holland)، الذي كان قد بدأ العمل عليها منذ بداية الستينيات، وكان هدفه تطور فهم إجرائية التطور الطبيعية وتصميم نظم صناعية لها مميزات مشابهة للنظم الطبيعية. تابع جون هولاند تطوير فكرته مع تلامذته وزملائه، ما قاده في عام 1975 إلى تأليف كتابه الأول عن الخوارزميات الجينية باسم "التكيف الطبيعي والنظم الذكية" (Adaption in Natural and Artificial Systems). وفي عام 1992 استخدم (John Koza) الخوارزميات الجينية لتطوير برامج سمحت بإنجاز بعض الأعمال، وقد أطلق على طريقته اسم البرمجة الجينية (Genetic_Programming (GP)).

إن الأمثَلة المحلية بدلاً من الوصول للحل الأمثل العام هي مشكلة غالباً ما تقع فيها طرائق البحث التدرجية (Gradient Search Methods)، لكن الخوارزميات الجينية حلت هذه المشكلة بإدخال مفهوم الطفرة. وعموماً فإن الخوارزميات الجينية تميل لأن تكون مكلفة حسابياً وبالتالي زمنياً . وكما أن الدافع المستمر لتحسين أداء النظم الحسابية، جعل من الخوارزميات الجينية حلاً مغرياً وجذاباً من أجل حل بعض مسائل الأمثلة التي لم يكن من الممكن حلها بزمن معقول باستخدام بقية الطرائق التقليدية السائدة.

استطاعت الخوارزميات الجينية لاحقاً حل العديد من أشهر المسائل مثل:

1. النظم الديناميكية اللاخطية، من ناحية التنبؤ وتحليل البيانات.
2. تصميم الشبكات العصبونية، من ناحية الهيئة والأوزان.
3. المسار المنحني لقذيفة أو صاروخ يتحكم بنفسه، ويوجه نحو الهدف.
4. تطوير برامج (LISP) أو ما يسمى بالبرمجة الجينية.

5. التخطيط الإستراتيجي.
6. إيجاد شكل جزيئات البروتينات.
7. حل مشكلة رجل الأعمال المتنقل، ومسألة جدول المواعيد والمهام.
8. توابع توليد الصور.
9. إنتاج مواد كيميائية صناعية على الحاسوب قبل إنتاجها بالمختبر.
10. إجراء محاكاة للتجارب النووية على الحاسوب.

6-4- خلفية بيولوجية (Biology Background):

من المعلوم أن هناك اختلافاً كبيراً في الأشكال والأطوال بين أفراد النوع نفسه من الكائنات الحية، إلا أن هذه الأفراد تتشابه مع بعضها بصفات تميزها عن غيرها من الأنواع. فمن الملاحظ أن الأبناء يشبهون آبائهم في بعض الصفات ويختلفون عنهم في صفات أخرى. يدعى العلم الذي يدرس التشابه والاختلاف في الصفات عند الأحياء بعلم الوراثة، حيث يقسم إلى أربعة أقسام؛ هي:

1. المستوى الجزيئي:

تدرس من خلاله بنية وآلية عمل الحمضين النوويين (DNA) و(RNA) اللذان يشكلان المادة الوراثية عند الأحياء ، ويشرفان على تركيب البروتين في الخلايا.

2. المستوى الصبغي والخلوي:

تدرس من خلاله الصبغيات كحوامل للمادة الوراثية. نطلق على مجموع المادة الوراثية المحمولة على الصبغيات اسم الذخيرة الوراثية.

3. المستوى الفردي:

تدرس فيه آلية انتقال المورثات من جيل لآخر والعلاقات بين هذه المورثات والتفاعل فيما بينها.

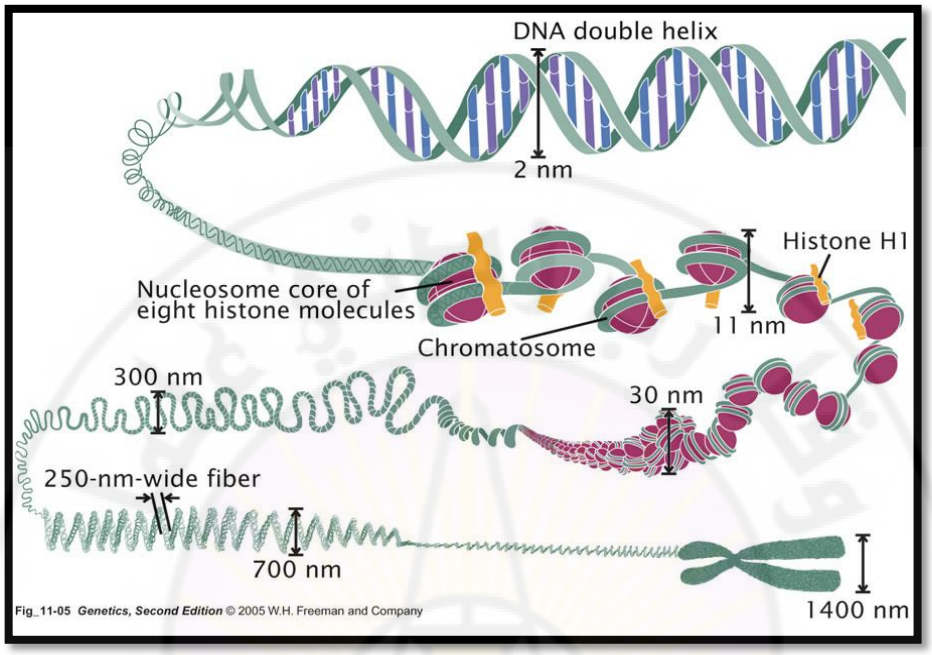
4. المستوى الجماعي:

تدرس فيه المورثات ضمن الجماعة من حيث ثباتها وتغيرها ، والشروط التي تؤدي إلى ثبات تواتر هذه المورثات، والعوامل المؤثرة، والتي تسبب تغيير هذا التواتر.

لوحظ من خلال التجارب على تلقيح الخلايا وانقسامها لتشكيل فرد جديد أن هناك تطابقاً في السلوكية بين المورثات والصبغيات. وبما أن عدد المورثات أكبر بكثير من عدد الصبغيات فقد افترض بأن المورثات تُحمل على الصبغيات وهذا ما يعبر عنه بالنظرية الصبغية التي تقول:

" تُحمل مورثات الصفات الوراثية على الصبغيات، وتنتقل عبرها من جيل إلى آخر".

وقد بينت التجارب أن لكل مورثة موقع محدد على الصبغي يقابله على الصبغي القرين موقع المورثة المقابلة لها. كما بينت النظرية الصبغية أن المورثات محمولة على صبغيات متواجدة في كل خلية من خلايا الكائن الحي. وكذلك بينت أن كلاً من الصبغي والمورثة يتكونان من حمض نووي هو (DNA)، حيث تتجمع المورثات التابعة للكائن الحي بشكل خطي لتشكل أحد صبغيات هذا الكائن ، وهذا ما يوضحه الشكل(6-2).



الشكل(6-2) توضيح أبعاد الصبغيات

6-5- نظرية داروين في التطور(Darwin Theory in Evolution):

تتبدل بعض صفات الأحياء ، لنجد حالياً أصنافاً من التفاح والقمح والبرتقال والدجاج والبقر والخيول ، لم تكن منتشرة في الماضي. وهي أصناف انحدرت من أصول برية عمل الإنسان على تربيتها واصطفاء المناسب له منها من حيث الثمار أو اللحم، ومن ثم إكثاره. كما تعمل الطفرات على تبديل صفات الكائن الحي مظهرة صفات وراثية جديدة، أي أنها تجعل الكائن الحي يتطور.

يميز الاخصائيون بين نوعين من التطور؛ هما:

1. تطور بسيط:

تتبدل بعض صفات النوع الواحد لكنه يبقى النوع نفسه، فالتفاح والقمح تطورا لكن التفاح بقي تفاحاً ، والقمح بقي قمحاً. وهذا التطور حدث للأحياء ، ولا يزال يحدث، وهو ثابت علمياً.

2. تطور كبير:

استرسل بعضهم ، وقالوا أن الأنواع الحية يتحول بعضها إلى بعض ، فالديدان تطورت إلى حشرات، والأسماك إلى ضفادع وهذه إلى ثدييات. وقد أُثبت خطأ هذا الرأي رغم وجود بعض المؤيدين له.

إن نظرية داروين من أهم نظريات التطور إذ تعتمد مبدأ الاصطفاء الطبيعي، ولها عدة بنود هي:

1) يوجد اختلاف كبير في الصفات بين أفراد النوع الواحد.
2) تتكاثر أفراد النوع الواحد وفق متوالية هندسية، فيزداد عددها كثيراً.
3) يحصل نتيجة ذلك تنافس بين أفراد النوع الواحد على الغذاء والمأوى وأماكن التكاثر.
4) يؤدي التنافس إلى حدوث عملية اصطفاء طبيعي تفرض بقاء الأفراد الأنسب، بحيث تموت الأفراد التي لا تملك الصفات المناسبة للبيئة أو تفشل في التكاثر، فتحذف صفاتها من الصفات الموجودة في الجماعة، وباستمرار هذا الاصطفاء خلال أجيال متتابعة يبقى الأفراد الأكثر تكيفاً مع البيئة.

هناك اعتراضات كثيرة على هذه النظرية منها:

1) يؤدي الاصطفاء إلى فرز الأفراد الحاملة لمورثات معينة ، فهو لا يخلق أنواعاً جديدة، بل يثبت النوع نفسه.

2) لا يعمل الاصطفاء الطبيعي بأسلوب تصنيفي من البسيط إلى المعقد، لأن استمرار الفرد بالبقاء قد يكون عائداً للصدفة.

3) لا تعد الطفرة جواباً على التكيف لأنها أمر احتمالي عشوائي، لكنها قد تبدي تكيفاً بعد حدوثها.

4) الاصطفاء الطبيعي يؤدي إلى تغير بسيط في الصفات أي تطور صغير فقط، ولا يؤدي إلى تطور كبير.

وبالتالي يمكن القول إن جميع الصفات الوراثية للكائن الحي والتي تحدده تماماً، تتواجد كمورثات (Genes) في كل خلية من خلاياه وتُحمل هذه الصفات أو المورثات على الصبغيات (Chromosomes). يتألف الصبغي من جينات وكل جين يتألف من عدة وحدات من حمض نووي هو (DNA)، وكل جين يُرمِّز بروتيناً خاصاً، ويُرمِّز سمة معينة كلون العينين، ولكل جين موضع محدد على الصبغي يسمى موضع الجين. تسمى المجموعة الكاملة للمادة الجينية أي جميع الصبغيات بمجموعة العوامل الوراثية. فيما تسمى كل مجموعة خاصة من الجينات ضمن المجموعة الكاملة بالنمط الوراثي (Genotype) الذي يعد منذ الولادة الأساس للمظهر الموروث للكائن الحي، والذي يشمل الصفات أو الملامح الشخصية الفيزيائية والصحية للكائن الحي كلون العينين وشكلهما وشكل الأذنين والذكاء.

خلال عملية إعادة إنتاج الكائن الحي من أبويه عن طريق التزاوج، فإن هذا الكائن يمر أولاً بمرحلة العبور (Crossover) أو إعادة التركيب، وفيها تعبر

الجينات من أحد الأبوين للآخر، ثم تنضم إلى الجينات الأصلية لتعطي صبغيات (Chromosomes) جديدة كاملة. وخلال عملية العبور يمكن أن تتعرض الذرية الجديدة (New Offspring) لتعديل أو طفرة (Mutation) في إحدى جيناتها. إن عملية الطفرة تعني أن أحد العناصر الأساسية في (DNA) قد تبدلت، والتي هي (C,T,G,A)، ويحدث هذا الخطأ بشكل أساسي حين يتم نسخ الجينات من الأبوين. إن ملاءمة (Fitness) الكائن الحي الجديد للظروف المحيطة به وتأقلمه معها يعد معيار النجاح في هذه العملية.

6-6- توصيف الخوارزميات الجينية (Basic Description):

تعد الخوارزميات الجينية مستلهمة من نظرية داروين في التطور، حيث يعتمد إيجاد حل لمشكلة ما بطريقة الخوارزميات الجينية على تطور عملية إيجاد الحل أو تطور الحل. وهي ممثلة للاعتقاد السائد بأن الذكاء البشري يخلق مع الإنسان، ويتم اكتسابه عن طريق الوراثة بشكل كبير. ومن هنا كانت الخوارزميات الجينية عبارة عن محاكاة لعملية التزاوج بين الكائنات الحية من النوع نفسه.

تبدأ الخوارزميات الجينية بمجموعة من الحلول تُمثَّل وكأنها صبغيات، وتسمى أفراد الجيل. تؤخذ الحلول (الصبغيات) من هذا الجيل وتستخدم لإيجاد نسل جديد (جيل جديد)، حيث يتوقع أن يكون هذا النسل أفضل من آبائه.

إن الحلول التي تنتخب لتكوين أفراد الجيل الجديد أو الذرية تختار عن طريق ملاءمتها للحل، وتملك الصبغيات ذات الملاءمة الأفضل (الأكبر أحياناً، والأصغر أحياناً أخرى)، فرصاً أكبر بأن تستخدم في إعادة إنتاج الأجيال اللاحقة. يُكرر هذا العمل حتى تتحقق شروطاً معينة (الوصول إلى أفضل حل أو إلى عدد محدد من الأجيال ...). تحتاج الخوارزميات الجينية قبل تطبيقها إلى معرفة تامة

بالنظام لتحديد البارامترات الهامة له. لذلك سوف نورد مبدئياً بعض النقاط الواجب معرفتها، في أثناء تطبيق الخوارزميات الجينية:

1. تابع الملاءمة (Fitness Function):

إن إيجاد تابع الملاءمة المناسب من أهم النقاط في الخوارزميات الجينية، حيث يختلف من مسألة إلى أخرى، ولا يوجد طريقة محددة لإيجاده، إنما يتبع للمصمم الذي يبحث عن حل لمسألته. يعبر هذا التابع عن مقدار التقارب بين الحل الأمثل والحل الذي تم إيجاده، وكلما كان الخطأ أقل كلما كان الحل أقرب للحل الأمثل.

2. الترميز (Encoding):

يجب ترميز المتحولات عند حل المشكلة بطريقة الخوارزميات الجينية، وقد يكون هذا الترميز ثنائياً أو حقيقياً أو محرفياً. تختلف طريقة الترميز حسب نوع المسألة، حيث يوجد طريقتان مختلفتان للترميز تعتمد على طبيعة المتحولات، إما متحولات مستمرة أو متقطعة. تعد مسألة المتحولات المتقطعة سهلة وواضحة في عملية الترميز على عكس المتحولات المستمرة التي تحتاج إلى بيتات (bits) أكثر لتعريف مجال واسع من القيم.

3. معاملات الخوارزمية الجينية:

تستخدم مجموعة من المعاملات في الخوارزميات الجينية بشكل متسلسل مؤدية للوصول إلى الحل الأمثل ، وبما يحاكي العملية البيولوجية، مثل الانتخاب والعبور والطفرة.

تجري عمليتا العبور والطفرة وفق نسبة احتمالية محددة خلال الحل ، وهي مختلفة من مسألة لأخرى. أما عملية الانتخاب فتتم وفقاً لمجموعة من الخوارزميات التي تأخذ بعين الاعتبار ملاءمة كل حل لإقرار مدى جودة استخدامه

في عملية التزاوج للجيل اللاحق. كثيراً ما تستخدم خوارزمية العجلة المتدحرجة التي توزع الحلول على قطاعات دائرية بنسبة توافق ملاءمة كل حل، وبالتالي يأخذ الحل الأفضل مساحة قطاعية أكبر ، ويزداد احتمال انتخابه للجيل اللاحق ، وتخفيض احتمال نقل الحلول الأسوأ إلى الجيل الجديد الذي يتم الحصول عليه بعد القيام بعملية العبور بين الحلول المنتخبة ، بهدف الوصول لحلول تتمتع بخواص أفضل من الآباء. ومن ثم تطبق عملية الطفرة مسببة توزيع فضاء الحل بشكل أكبر لتجنب الوقوع في منطقة حلول محددة، كما يحدث لو أردنا إيجاد النهاية المحلية العظمى لتابع يحوي عدة نهايات عظمى محلية. بمعنى آخر، البحث عن مناطق جديدة ضمن فضاء الحل.

4. حجم الجيل (Population Size):

يعرف حجم الجيل بأنه عدد الحلول ضمن الجيل الواحد. إن الزيادة في حجم الجيل تعني زيادة في فرصة إيجاد الحل الأمثل. فكلما زاد عدد النقاط التي ننطلق منها في فضاء الحل، كلما زادت حدة اقترابنا من الحل المطلوب.

5. خطوات إيجاد الحل:

اعتماداً على النقاط السابقة يمكن إتباع الخطوات المبينة لإيجاد الحل الأمثل المطلوب:

1) تحديد القيم الابتدائية للخوارزمية كحجم الجيل واحتمال الطفرة وبقية البارامترات اللازمة للخوارزمية.
2) توليد جيل ابتدائي بشكل عشوائي، وبحجم جيل محدد سلفاً.
3) تطبيق خوارزمية الانتخاب المناسبة للحصول على أزواج من الحلول ، وبحيث يكون عدد الأزواج مساوياً إلى نصف حجم الجيل.

4) تطبيق خوارزمية العبور بين الأزواج المنتخبة للوصول إلى الجيل الجديد ذي الحجم المحدد سلفاً.

5) إحداث طفرة على الجيل الجديد باحتمال محدد.

6) استبدال الجيل الجديد بالجيل القديم.

7) فحص شرط إنهاء الخوارزمية ، سواء أكان بعدد مرات التكرار أ م بزمن التنفيذ أم بدقة الحل، فإن لم يتحقق، فالعودة إلى الخطوة الثالثة، وإلا إنهاء الخوارزمية وإظهار الحل الأمثل الذي يتمتع بأفضل مستوى ملاءمة.

6-7- فضاء البحث (Search Space):

عند حل مشكلة ما وإيجاد بعض الحلول، فغالباً ما يكون بعضها أفضل من بعض. نسمي الفضاء المكون من جميع الحلول المقبولة بفضاء البحث أو فضاء الحالة، وكل حل مقبول يمكن أن يمثل بنقطة حسب ملاءمته ليكون حلاً للمشكلة. نستطيع باستخدام الخوارزميات الجينية البحث عن أفضل حل من بين كثير من الحلول المقبولة، والذي يمكن تمثيله بنقطة واحدة في فضاء البحث.

من المعلوم أن أفضل حل يكون عادة على الحدود الخارجية لفضاء البحث فمثلاً: تدمير طائرة معادية يجب أن يتم بأقل زمن وبأقل ذخيرة ممكنة. كذلك يجب أن يتم مسح أكبر منطقة ممكنة من قمر صناعي عندما يمر فوقها. أي أننا دوماً نبحث إما عن قيمة صغرى أو عن قيمة عظمى في فضاء البحث. مع مرور الزمن يصبح فضاء البحث محدداً ومعرفاً بشكل أفضل، لكن عادة نعرف عدة نقاط منه فقط عند البدء. وخلال عملية الحل باستخدام الخوارزميات الجينية

يتم إيجاد وتوليد حلول أخرى تمثل أيضاً نقاط حلول ممكنة ، وتعتبر كتطور للحل أي (Evolution proceeds). لكن هناك مشكلة أساسية تتجلى في تحديد مكان البحث وبدايته، خاصة عندما يكون البحث معقداً وشائكاً.

يوجد عدة طرائق للبحث عن حل مناسب لكن ليس من الضروري أن تقدم جميعها الحل الأمثل دائماً، من هذه الطرائق:

1. طريقة تسلق الهضبة (Hill Climbing).
2. البحث بطريقة تابو (Tabu Search).
3. الخوارزميات الجينية (Genetic Algorithms).

بعد إيجاد الحلول بهذه الطرائق يتم تقييم النتائج للوصول لحلول جيدة ، حيث لا توجد طرائق أخرى لمعرفة ما هو الحل الأمثل.

أحد الأمثلة على صنف المشاكل التي لا يمكن حلها بالطرائق التقليدية هي ما يسمى (NP–Problems). يمكن حل العديد من المشاكل بتطبيق خوارزميات سريعة ومتعددة الأطراف (المتحولات أو الحدود). كما أن هناك بعض المشاكل التي لا يمكن حلها باستخدام هذه الخوارزميات. هناك مشاكل هامة وعديدة من الصعب جداً إيجاد حل لها، لكن من السهل جداً التحقق من حلٍ ما أنَّهُ لها أو لا، وهذا ما يقود إلى (NP–complete problems). تعتمد (NP) على كونها متعددة الأطراف (المتحولات) إلا أن هذه المتحولات غير معرفة تماماً وهذا ما يسمى (Nondeterministic Polynomial) (NP)، وهذا يعني أنه من الممكن أن نخمن (Guess) الحل عن طريق خوارزميات غير معرفة تماماً أي تعتمد العشوائية ومن ثم نختبر الحل. أي أننا نحاول إيجاد الحل بهذه الطرائق شرط أن يكون زمن إيجاده معقولاً. إن دراسة المشاكل الكاملة متعددة المتحولات، والتي تكون فيها هذه المتحولات غير محددة تماماً، تقتصر على المشاكل التي

يكون جوابها نعم أو لا، لأن هناك أعما لاً مركبة المخارج وهذا الصنف يسمى مشاكل (NP–Hard). إن من صفات المشاكل (NP) هي سهولة الخوارزميات ، حيث نستطيع من لمحة واحدة الوصول إلى حلول مفيدة، لكن تبقى مشكلة محاولة إيجاد كل الحلول الممكنة أو أفضل حل منها، عندها تصبح عملية إيجادها بطيئة جداً. إحدى طرائق حل المشاكل (NP) هي الخوارزميات الجينية وإحدى المشاكل التي قامت بحلها الخوارزميات الجينية هي مشكلة التبضع من عدة محلات.

6–8– الخطوط العامة للخوارزميات الجينية (Outline of Genetic Algorithms):

يوضح الشكل(6–3) المخطط التدفقي العام للخوارزميات الجينية الذي نوجز مراحله بما يأتي:

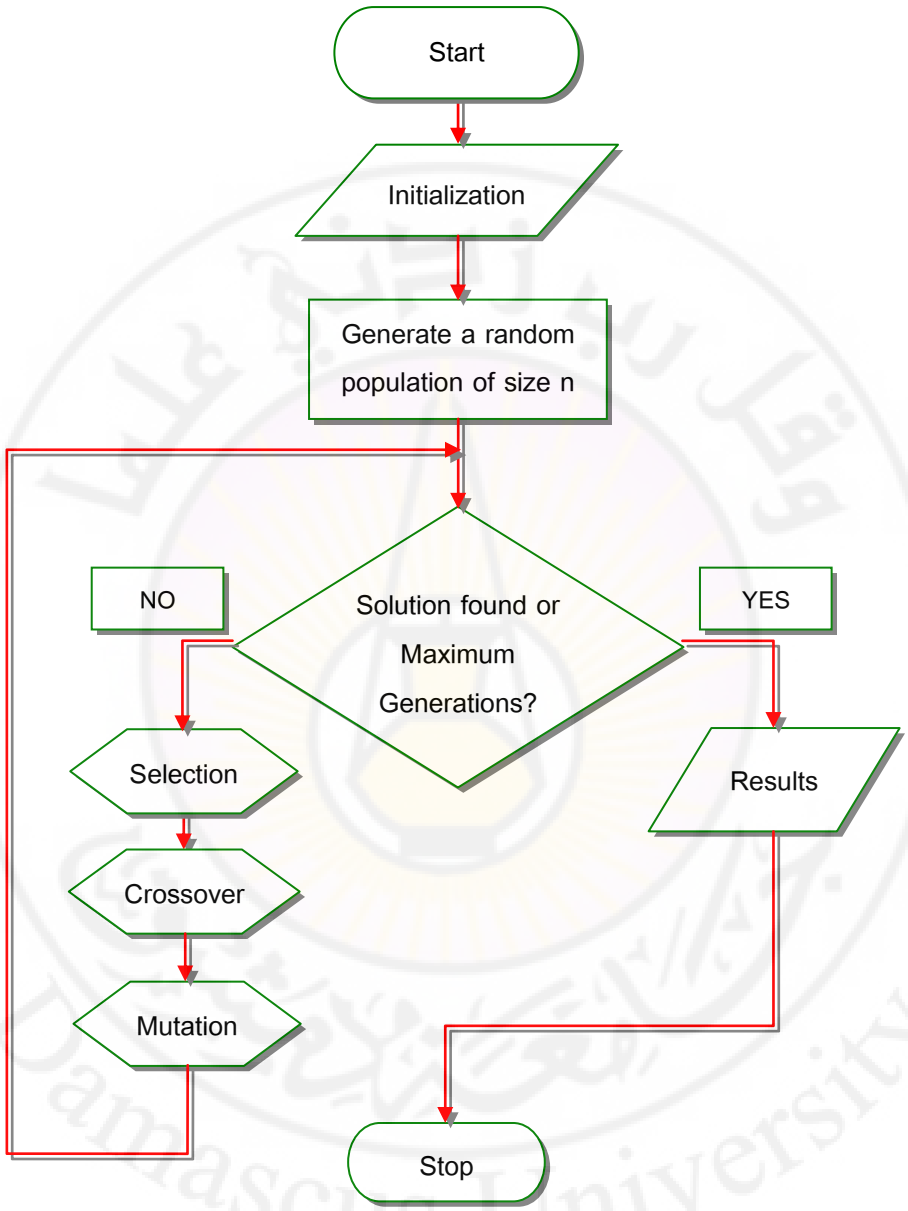
1. البدء (Start):

بدء تنفيذ الخوارزمية الجينية.

2. التأهيل الابتدائي (Initialization):

تحديد بعض المعايير والثوابت وقيم المتغيرات حسب رغبة المستخدم مثل:

1) حجم الجيل (Population Size): يقصد به عدد الأفراد ضمن الجيل، وهو من العوامل الهامة التي يتوقف عليها أداء الخوارزمية. بشكل عام، فإن زيادة عدد الأفراد إلى درجة كبيرة سوف يجعل زمن الحصول على النتائج كبيراً، مما يقلل من فائدة هذه النتائج. فعلى سبيل المثال، ما الفائدة من حصولنا على احتمال عالٍ في حدوث كارثة طبيعية أو صناعية بعد فوات الأوان؟. بالمقابل فإن زيادة حجم الجيل سوف يقلل من الخطأ المرتكب في النتيجة. أيضاً هناك مشكلة



الشكل(6-3) المخطط التدفقي للخوارزميات الجينية

عند كون حجم الجيل صغيراً إذ تنخفض دقة النتائج بشكل ملحوظ، لأنه لن يغطي كامل المجال المدروس بشكل جيد. بينت بعض البحوث أن اختيار حجم الجيل (30) هو اختيار موفق للعديد من المسائل.

2) احتمال الطفرة: بالطبع نسمع عن الطفرة البيولوجية بشكل متكرر بين الآونة والأخرى، ولكنها تبقى ضمن حدود احتمالية متدنية. أيضاً هنا نختار احتمال الطفرة في الخوارزميات الجينية بقيمة احتمالية متدنية.

3) عدد مرات التكرار: تكرر الخوارزمية عدداً محدداً من المرات للوصول إلى مستوى جيل جيد، وهنا نحدد أكبر عدد لمرات التكرار.

3. توليد الجيل الابتدائي:

كل مجتمع يبدأ من جيل، وهذا الجيل تجمعه صفات عامة مشتركة. وهذا ما نطبقه في خوارزميتنا ، حيث يولد جيل عشوائي من الأفراد للانطلاق منه والحصول على بقية الأجيال، وبحيث يكون عدد أفراد الجيل (n) مساوياً لحجم الجيل الذي اختير في أثناء التأهيل الابتدائي. إن كل فرد من أفراد الجيل الابتدائي تم توليده بشكل عشوائي وهو يمثل حلاً مقبولاً للمسألة وبلا شك، فإنه ينتمي إلى فضاء البحث المحدد من نص المسألة وفق قيود زمنية أو مكانية، وبالتالي يكون لدينا (n) صبغياً عشوائياً، وكل صبغي يمثل عينة أو فرد في المجتمع ، وبالتالي يكون لدينا مجتمع يحتوي (n) فرداً.

4. دراسة الملاءمة (Fitness):

نختار تابع الملائمة F(x) المناسب للمسألة، ومن ثم نحسب قيمته لجميع الأفراد أو الصبغيات لمعرفة تقارب هذه العينات من الحل الأمثل.

5. الذرية الجديدة (New Population):

تولد الذرية الجديدة أو الجيل الجديد بتكرار الخطوات الآتية حتى تكتمل الذرية:

1) الانتخاب أو الاختيار (Selection):

نختار أبوين (اثنين من الصبغيات) من الذرية السابقة ، حسب قيمة تابع الملاءمة الموافق لهما ، حيث أنه كلما كانت درجة ملاءمة الصبغي أفضل، كلما كانت فرصة انتقائه أكبر ، وهذا يوافق قانون الانتقاء الطبيعي في نظرية داروين للتطور. تنفذ هذه العملية باختيار مناسب لخوارزمية الانتخاب، حيث يوجد العديد من الخوارزميات التي يمكن للمستخدم اختيار إحداها في مرحلة التأهيل الابتدائي.

2) عملية العبور (Crossover):

يجري تبادل عدة جينات بين الأبوين مما ينتج عنه ذرية جديدة (أبناء) ، أما في حال عدم عبور أي جين من الأبوين ، فتكون للأبناء صبغيات الآباء نفسها. وهذا يوافق عملية التزاوج بين الكائنات من النوع نفسه، حيث تكون الصفات الوراثية للأبناء مؤلفة من قسمين، كل منهما قادم من أحد الأبوين. أيضاً هنا، تنفذ العملية باختيار مناسب لخوارزمية العبور، حيث يوجد العديد من الخوارزميات التي يمكن للمستخدم اختيار إحداها في مرحلة التأهيل الابتدائي.

3) الطفرة (Mutation):

تحدث حسب احتمالية التطور ، حيث تؤثر هذه العملية على الجينات فتغيرها دون تغيير مكانها في الصبغي نسبياً، وهذا يوافق الطفرة البيولوجية التي تنتج صفات وراثية جديدة . تختلف طريقة تنفيذ الطفرة حسب طريقة الترميز المعتمدة في حل المسألة، فلكل طريقة ترميز مختلفة، طريقة توليد طفرة مختلفة.

4) التبديل (Replace):

استبدال الجيل الجديد المولد بالجيل القديم الذي بدأنا به بعد فحص شرط التوقف، ومعاودة فحص شرط التوقف على الجيل الجديد، وعند عدم تحققه، إعادة تكرار بعض خطوات الخوارزمية مجدداً من انتخاب فعبور فطفرة.

6. شرط التوقف:

إن شرط توقف الخوارزمية يلعب دوراً هاماً في النتيجة النهائية ، وفي كلفة العملية عموماً ، وهو شرط يقرره مستخدم النظام البرمجي في مرحلة التأهيل الابتدائي . تجري عملية فحص على أفراد الجيل الجديد ، فإذا كان أحد الحلول محققاً أو مُرضِياً كحل للمسألة ، عندها نتوقف عن تنفيذ الخوارزمية ونحدد الحل الأمثل. أما عند عدم تحقق شرط الخروج من الخوارزمية ، أو كون الحل ليس مثالياً، فإننا نتابع تكرار بعض الخطوات السابقة على أفراد الجيل الجديد.

يوجد العديد من شروط التوقف الممكنة التي يمكن للمبرمج إدخالها وعرضها للمستخدم كي يختار منها في مرحلة التأهيل الابتدائي، ومن هذه الشروط:

1) قيمة تابع الملاءمة:

من الممكن أن يكون شرط التوقف هو الوصول بالنتيجة إلى قيمة خطأ محددة مسبقاً نسبة للقيمة المرغوبة. فكما هو معلوم، نسعى أحياناً إلى أن يكون تابع الملاءمة ينتهي إلى الصفر عند الحل الأمثل، وبالتالي نحدد شرط التوقف كقيمة قريبة من الصفر، وكلما كانت القيمة أقرب للصفر، كلما اقترب الحل من الحل الأمثل.

2) زمن التنفيذ:

يمكن أن يختار المستخدم شرطاً زمنياً للتوقف، كاشتراط عدم تجاوز زمن التنفيذ لقيمة زمنية محددة، وأخذ أفضل حل تم الوصول له خلال فترة التنفيذ . يفضل استخدام هذا الشرط في أنظمة الزمن الحقيقي حيث يشترط الحصول على النتيجة قبل وصول المهمة إلى زمنها الحرج ، والذي يعرف بأنه آخر لحظة زمنية يمكن قبول انتهاء تنفيذ المهمة عندها. بعض خوارزميات جدولة مهام الزمن الحقيقي تعتمد على الحصول على أقصى دقة للمهمة بحيث لا تتجاوز زمنها الحرج.

3) عدد مرات التكرار:

من شروط التوقف الممكنة هو اختيار العدد الأعظمي للأجيال، أي عدد مرات تكرار الخوارزمية. نتوقع أن يزداد الاقتراب من الحل الأمثل بزيادة عدد الأجيال، ولكن لا يمكن الاستمرار في عملية التكرار إلى الحد الذي تنخفض فيه الفائدة من الحل. يمكن للمستخدم تحديد عدد التكرارات الممكنة بقيمة أولية، ومن ثم إعادة تقدير القيمة المناسبة في التنفيذ التالي للخوارزمية.

يدمج أكثر من شرط توقف في النظام البرمجي الخاص بالخوارزميات الجينية بحيث يتسنى للمستخدم اختيار شرط توقف محدد فقط، أو اختيار أكثر من شرط وفق علاقة منطقية (علاقة "و" أو علاقة "أو"). فمثلاً يختار المستخدم شرط التوقف بالشكل الآتي:

زمن التنفيذ (25 Sec)، وعدد مرات التكرار (1000).

في هذه الحالة تتوقف الخوارزمية عند تحقق الشرطين سوية، أي تجاوز زمن التنفيذ للقيمة (25 Sec)، وتجاوز عدد مرات التكرار للقيمة (1000).

6-9- ترميز الصبغي (Choromosome Encoding):

يجب ترميز المتحولات عند حل المسائل بطريقة الخوارزميات الجينية، وقد يكون هذا الترميز ثنائياً أو حقيقياً أو محرفياً حسب نوع المسألة. هناك طريقتان مختلفتان للترميز تعتمدان على طبيعة المتحولات، فإما أن تكون المتحولات مستمرة أو متقطعة، وتعد مسألة المتحولات المتقطعة سهلة وواضحة في عملية الترميز ، على عكس المتحولات المستمرة التي تحتاج إلى (bits) أكثر للعمل على تعريف مجال قيم غير محدد (غير منتهٍ)، وسنناقش حالتي الترميز بشيء من التفصيل.

1. المتحولات المتقطعة (Discrete Variables):

تعرِّف المتحولات المتقطعة مجالاً من القيم الصحيحة، مثل القيم الصحيحة من (0) إلى (100)، أو مجموعة قيم محددة كعناصر المجموعة {3,5,6,11,34,65,73,121}. من المستحسن في هذه الحالة تخزين القيم في مصفوفة أو جدول وترق يم عناصر المصفوفة لنحصل على الترميز، أي نرمز العنصر الأول بالرمز (1) ثم الثاني بالرمز (2) وهكذا حتى الرمز (n)، وبالتالي يمكن ترميز القيم مباشرة بالترميز الثنائي باستخدام طرائق التحويل بين النظام العشري والثنائي. يجب الملاحظة هنا أن طول سلسلة الترميز لعناصر المصفوفة تحسب من أساس العدد (2) أي n(2)، وبالتالي قد تكون تساوي عدد القيم المراد ترميزها أو أكبر منها. فعند ترميز الأرقام من (1) إلى (100) يكون طول السلسلة سبع خانات ثنائية ، حيث 128=7(2)، أي تكون السلسلة قادرة على ترميز (128) قيمة، مما يضعنا في إشكالية جديدة، وهي كيف ية الترميز عند تجاوز خانات الترميز لقيم المسألة؟

في الحقيقة هناك طريقتان لحل هذه المشكلة، الأولى هي طريقة البتر (Truncation)، والثانية هي طريقة الاستيفاء (الاستكمال) (Interpolation).

1) طريقة البتر (Truncation):

في هذه الطريقة فإن جميع القيم خارج المجال الذي نحتاج له تسند إلى آخر قيمة في مجموعة القيم المراد ترميزها . فإذا عدنا للمثال السابق فإن الرموز من (100) إلى (127) تعبر عن القيمة (100)، وهذا بالتالي يجعل احتمالية حدوث القيمة (100) أعلى بكثير من احتمالية حدوث أي قيمة أخرى في المجموعة، ما يسيء لخوارزمية الانتخاب. تمتاز هذه الطريقة بالبساطة وسهولة البرمجة.

2) طريقة الاستيفاء (Interpolation):

تستخدم هذه الطريقة لحل إشكالية طريقة البتر بعدم تساوي احتمالية حدوث القيم، وذلك بتحقيق احتمال متساوٍ لحدوث جميع القيم. يتم هنا تقسيم مجال أرقام الترميز على مجال القيم المراد ترميزها . فبفرض وجود (300) متحول يراد ترميزها ، عندها يكون طول سلسلة الترميز (9)، أي يوجد عدد من الرموز هو $512=(2)^9$، لذلك يتم تقسيم عدد الرموز على المتحولات بالشكل:

$$512/300 = 1.707$$

وبالتالي المجال [1.707 , 0] يرمز القيمة 1.

المجال [3.414 , 1.707[يرمز القيمة 2.

المجال [5.121 , 3.414[يرمز القيمة 3 وهكذا.

وبذلك يصبح احتمال حدوث أي متحول من مجموعة المتحولات متساوٍ، ويساوي في هذه الحالة $1.707/512=1/300$. تمتاز هذه الطريقة بأنها أفضل من طريقة البتر من ناحية تساوي احتمال حدوث المتحولات تقريباً، لكنها أكثر تعقيداً ومتطلبات برمجية.

2. المتحولات المستمرة (Continuous Variables):

تعد المشكلة الرئيسة في تعريف المتحولات المستمرة هي وجود عدد لا نهائي من القيم بين أي مجالين صغيرين، فالمجال [1.01 ,1.00] مثلاً، يحتوي على عدد لا نهائي من القيم الحقيقية. يوجد طريقتان لحل هذه المشكلة؛ هما:

1) الطريقة الأولى:

تحدد درجة سماحية للقيمة (Precision) منذ البداية، أي يحدد عدد الخانات بعد الفاصلة، فإذا كانت درجة السماحية للقيم ضمن المجال [0, 10] هي (3) أرقام بعد الفاصلة، فإن عدد القيم المراد ترميزها سيساوي:

$$[(10-0)\times 10^3+1]=10001$$

وبالتالي نحتاج إلى سلسلة ترميز مكونة من (14) خانة، وبالتالي القدرة على ترميز $16384=^{14}(2)$ قيمة، وهنا ظهرت المشكلة السابقة نفسها، والتي يمكن حلها عن طريق إجرائية الاستيفاء.

2) الطريقة الثانية:

يتم تحديد عدد خانات الترميز مسبقاً، ثم يحدد حجم الخطوة الموافقة لها، فبفرض أن x_i^u ، x_i^l تعبران عن الحدود العليا والدنيا للمتحول x_i، وكان D_i^{max} تعبر عن أعلى رقم يمكن ترميزه حسب عدد الخانات المختارة، فإن العلاقة الآتية:

$$S_i=\frac{(x_i^u-x_i^l)}{D_i^{max}}$$

تعطي حجم الخطوة S_i للمتحول x_i.

عند ترميز أي قيمة بترميز ما (نظام ثنائي مثلاً) فإن هذا الترميز يطلق عليه اسم الجين (gene) وكل جين يتألف من صفات وراثية (Alleles) (0 أو 1 في النظام الثنائي)، وعند جمع سلسلة ترميزات تعبر عن ترميز حلول المسألة

عندها تسمى هذه السلسلة بالصبغي (Chromosome)، أي أن كل صبغي يتألف من جين واحد على الأقل أو عدة جينات.

إن أول سؤال يطرح عند حل مسألة ما هو كيف سنقوم بترميز الجينات ومن ثم الصبغيات. إن عملية الترميز تعتمد على طبيعة المسألة، إذ يوجد عدة طرائق مستخدمة وناجحة للترميز ، حيث سنتعرف على أهم تلك الطرائق في الفقرات اللاحقة.

6-10- طرائق الترميز (Encoding Type):

تتوافر العديد من طرائق الترميز المستخدمة في الخوارزميات الجينية ، كالترميز الثنائي والعشري والقيمة والشجري.

1. الترميز الثنائي (Binary Encoding):

استخدمت البحوث الأولى في الخوارزميات الجينية هذا النوع من الترميز بسبب سهولة التعامل معه، ما جعله أكثر طرائق الترميز استخداماً. يكون هنا كل صبغي عبارة عن شريط رقمي مؤلف من تتابع من واحدات وأصفار كما في الشكل(6-4)، الذي يبين ثلاثة صبغيات مرمزة ثنائياً لثلاث مسائل مختلفة، وهذا ما يبرر اختلاف أطوال الصبغيات الثلاثة حسب المسألة، حيث أنه للمسألة الواحدة تكون جميع الصبغيات بالطول نفسه:

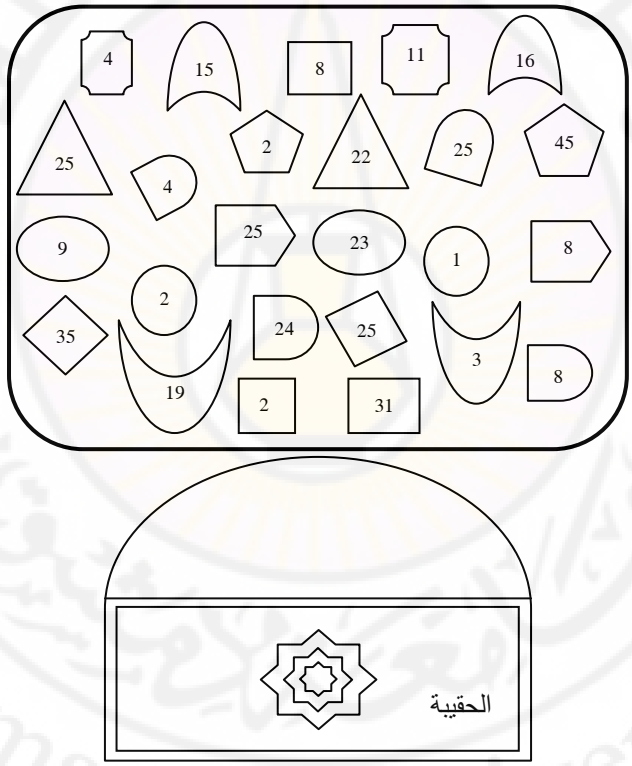
Chromosome A	0 1 1 1 0 0 0 1 0 1
Chromosome B	1 0 1 0 1 1 1 0 1 0 1 0
Chromosome C	1 1 1 1 1 1 1 0 1 0 1 0 1 1

الشكل(6-4) ترميز الصبغيات ثنائياً

يعطي الترميز الثنائي صبغيات عديدة محتملة، حتى بالنسبة لعدد محدود من الصفات الوراثية.

مثال: الحقيبة النسائية أو حقيبة الظهر (Knapsack Problem).

هي إحدى المسائل التي تحل باستخدام الخوارزميات الجينية بترميز الصبغيات بشكل ثنائي. يوضح الشكل(6-5) تمثيلاً مبسطاً للمسألة مع توضيح لقيم الأغراض:



الشكل(6-5) تمثيلاً مبسطاً لمسألة الحقيبة النسائية

وصف المسألة:

ذهبت إحدى السيدات إلى التسوق، وهي تحمل حقيبة ذات حجم معلوم. يوجد مجموعة متنوعة من الأغراض ذات قيم وأحجام مختلفة ، والمطلوب تحديد الأغراض التي تجعل قيمة الحقيبة أعلى ما يمكن.

بالطبع، يجب أن لا يزيد حجم مجموع الأغراض عن حجم الحقيبة. كما أن قيمة الغرض مستقلة عن حجمه، فمن الممكن أن يكون حجم الغرض كبيراً وقيمته كبيرة، وغرض آخر بالحجم نفسه لكن بقيمة منخفضة.

الترميز:

يمكن حل المسألة بافتراض أن الحل أو الصبغي عبارة عن سلسلة من الأرقام الثنائية بطول يساوي عدد الأغراض المتوافرة، والتي يمكن إدخالها في الحقيبة. كل رقم ثنائي يعبر عن غرض ما، وبحيث تكون قيمته الواحد عند اختيار وجود الغرض ضمن الحقيبة، وصفراً عند عدم اختيار الغرض.

فإذا كان الصبغي بالشكل الآتي:

Chromosome A = **[1001110001010001001011101]**

فهذا يعني أنه لدينا (25) غرضاً يمكن الاختيار من ضمنهم للتواجد ضمن الحقيبة. وقد قامت السيدة (من اليسار) بوضع الغرض الأول والرابع والخامس والسادس والعاشر و...الخ ضمن الحقيبة، مع تحقق شرط أن مجموع أحجام الأغراض المختارة أقل أو يساوي حجم الحقيبة. ليست كل الاحتمالات الممكنة للصبغي ستكون منتمية إلى فضاء البحث، فبعض الاحتمالات سوف تكون بمجموع أحجام أكبر من حجم الحقيبة، وبالتالي لن يكون هذا الاحتمال حلاً مقبولاً. عند انتماء الصبغي لفضاء البحث، يحسب تابع الملاءمة بجمع قيم الأغراض الموجودة ضمن الحقيبة أي ذات القيمة واحد في الصبغي، وكلما كانت

قيمة تابع الملاءمة أكبر، كلما اقتربنا من الحل الأمثل، الذي يحقق أعلى قيمة للحقيبة.

2. الترميز التبديلي (Permutation Encoding):

يفيد الترميز التبديلي (الطبيعي أو العشري) في المشكلات التي يكون مضمونها عمليات فرز، مثل مشكلة رجل المبيعات المتنقل أو مشكلة ترتيب المهام. في هذه الطريقة كل صبغي هو عبارة عن شريط محرفي (String) يمثل الأماكن بشكل متسلسل كما في الشكل(6-6) الذي يبين ثلاثة صبغيات مرمزة عشرياً لثلاث مسائل مختلفة:

Chromosome A	1 5 3 2 6 4
Chromosome B	6 5 8 7 2 1 3 4 9
Chromosome B	6 5 8 10 7 2 1 3 4 9

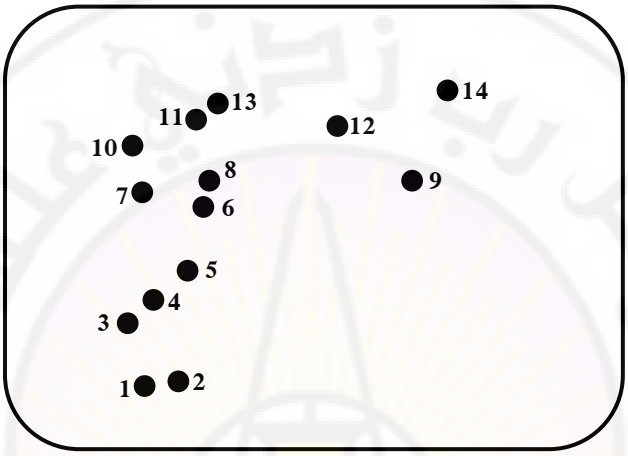
الشكل(6-6) ترميز الصبغيات بشكل عشري

إن هذه الطريقة مفيدة في المشكلات الترتيبية، لكن في بعض الأنواع من عمليات العبور أو عمليات التطور يجب الانتباه إلى انتماء الصبغي إلى فضاء الحل بتحقيق بعض التصحيحات لجعل الصبغي نظامياً ومناسباً (كمثال هل يملك هذا الصبغي تتابع حقيقي).

مثال: البائع المتجول.

هي إحدى المسائل المشهورة والمعيارية والمعقدة جداً، حيث يتطلب حلها استخدام الذكاء الصنعي كالخوارزميات الجينية، وذلك بترميز الصبغيات بشكل

عشري. يوضح الشكل(6-7) تمثيلاً مبسطاً للمسألة مع توضيح إحدى الحلول الممكنة في الشكل(6-8):



الشكل(6-7) تمثيلاً مبسطاً لمسألة البائع المتجول



الشكل(6-8) إحدى حلول مسألة البائع المتجول

وصف المسألة:

يرغب أحد الباعة المتجولين بعرض بضائعه للزبائن وذلك بالمرور عليهم في أماكن متفرقة (مجموعة مدن)، بحيث يحقق أقصر مسافة ودون المرور بالمكان إلا مرة واحدة فقط، مع مراعاة عدم وجود تقاطعات في خط سيره. على البائع الانطلاق من مكان ما والعودة إليه في نهاية الرحلة. وبالتالي فالمطلوب إيجاد تتابع الأماكن الواجب زيارتها وفق الشروط الموضوعة.

بالطبع ، يجب مراعاة المرور على جميع الأماكن، كما أن الطرق بين الأماكن وفق خط مستقيم مباشر ذي طول محدد.

يمكن زيادة تعقيد المسألة بإضافة بعض الشروط والمحددات. فمثلاً، يمكن وضع الزمن اللازم للانتقال بين الأماكن، أي بمعنى آخر السرعة القصوى الممكنة بين كل مكانين، وهي تتعلق بالتضاريس الموجودة. كما يمكن إضافة الكلفة المادية للانتقال بين كل مكانين. وكذا تعمم المسألة للأخذ بالاعتبار المسافة والزمن والكلفة المادية.

الترميز:

يمكن حل المسألة بافتراض أن الحل أو الصبغي عبارة عن سلسلة من الأعداد العشرية بطول يساوي عدد الأماكن المطلوب زيارتها. كل عدد عشري يعبر عن مكان ما، ويفضل أن تكون القيم متتابعة من الواحد وحتى آخر مكان. يصف الصبغي بالتالي ترتيب الأماكن التي سيزورها البائع المتجول.

فإذا كان الصبغي بالشكل الآتي:

Chromosome A = **[5 6 8 7 10 11 13 12 14 9 2 1 3 4]**

فهذا يعني أنه لدينا (14) مكاناً أو مدينة يجب المرور عليها. وقد قام البائع بالبدء من المدينة الخامسة فالسادسة فالثامنة وهكذا حتى المدينة الرابعة

حيث يفترض عودته إلى نقطة البداية. ليس من الضروري أن يكون هذا هو الحل الأمثل، ولكنه حل مقبول ينتمي إلى فضاء الحل ومهمة الخوارزمية الجينية إيجاد أفضل حل ضمن فضاء الحل. نلاحظ عدم تكرار أي عدد أكثر من مرة كما أن جميع الأعداد الصحيحة ضمن المجال [14 ,1] موجودة في الصبغي. عند انتماء الصبغي لفضاء البحث، يحسب تابع الملاءمة بجمع قيم المسافات بين كل مدينة والتي تليها مع مراعاة المسافة من آخر مدينة في الصبغي إلى المدينة الأولى فيه، وكلما كانت قيمة تابع الملاءمة أصغر، كلما اقتربنا من الحل الأمثل الذي يحقق أقصر مسافة انتقال.

3. ترميز القيمة (Value Encoding):

ترمز قيمة المتحول مباشرة بشكل مناسب. تستخدم هذه الطريقة في المشاكل التي تحوي أرقاماً صعبة مثل الأرقام الحقيقية، حيث يكون الترميز الثنائي عندها صعباً جداً. في هذه الطريقة كل صبغي عبارة عن تتابع لبعض القيم والتي يمكن أن تكون أي شيء مرتبط بالمشكلة كالأرقام الحقيقية أو الأحرف أو الكلمات كما في الشكل(6-9)، الذي يبين ثلاثة صبغيات مرمزة بالقيمة لثلاث مسائل مختلفة:

Chromosome A	2.282 7.524 2.457 0.379 9.954
Chromosome B	(back), (right), (back), (left), (forward)
Chromosome C	HDIFJDDLIERJFABDJEDFLFEGT

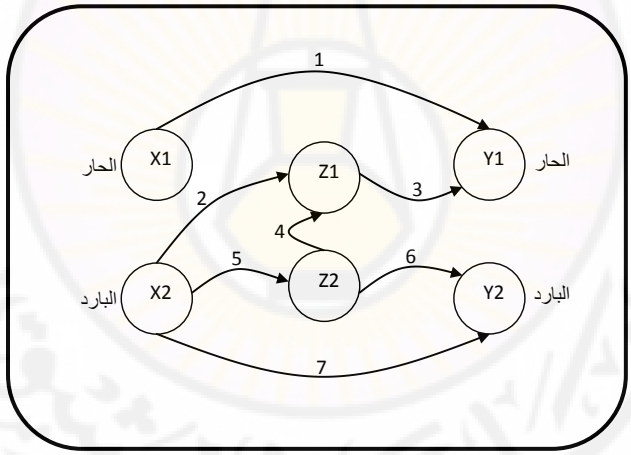
الشكل(6-9) ترميز الصبغيات بالقيمة

إن هذه الطريقة في الترميز هي الأكثر عمومية بين طرائق الترميز الأخرى.

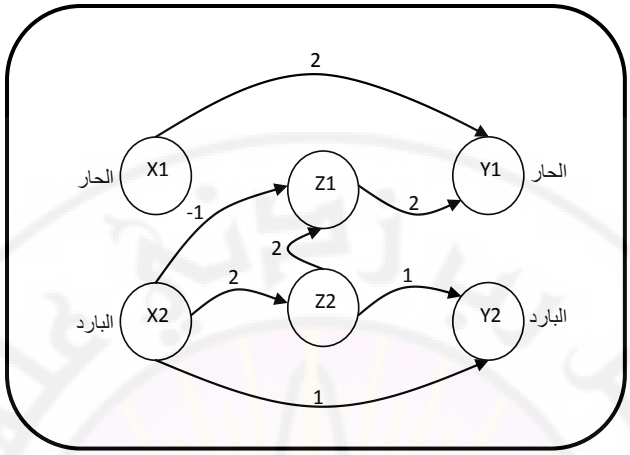
مثال: حساب أوزان الشبكة العصبونية.

إن أصعب مرحلة في حل المسائل المعقدة باستخدام الشبكات العصبونية هي مرحلة إيجاد االأوزان المناسبة للعصبونات، والتي تعطي أعلى دقة في حل المسألة. يمكن حل هذه المرحلة باستخدام الخوارزميات الجينية التي تقوم بإيجاد أفضل القيم لأوزان الشبكة العصبونية ، وبالتالي تكون المسألة الأساسية حلت باستخدام تقنيتي ذكاء صنعي هما الشبكات العصبونية والخوارزميات الجينية.

يوضح الشكل(6-10) تمثيلاً مبسطاً لشبكة عصبونية تحاكي عملية الإحساس بالمواد الحارة والباردة لدى البشر، مع ترقيم الترابطات بين العصبونات. كما يوضح الشكل(6-11) الأوزان المناسبة للمسألة.



الشكل(6-10) تمثيلاً مبسطاً لمسألة تحديد أوزان الشبكة العصبونية مع ترقيم الترابطات



الشكل(6-11) أوزان الشبكة العصبونية لمسألة المنبهات الحارة والباردة

وصف المسألة:

من المعلوم فيزيولوجياً أنه عند تطبيق منبه بارد على الإنسان لفترة قصيرة فإنه يستشعره وكأنه منبه حار، أما عند تطبيق منبه حار سواء أكان لفترة قصيرة أم طويلة فإن إدراكه للمنبه الحار يكون صحيحاً. تستطيع الجملة العصبية لدى الإنسان استشعار المنبهات الباردة بشكل صحيح فقط عند تطبيق هذا المنبه لفترة طويلة نسبياً. وبالتالي فالمطلوب تصميم شبكة عصبونية باستخدام شبكة McCulloch-Pitts لنمذجة عملية الإحساس للمنبهات الحارة والباردة لدى البشر.

نفترض أن هيكلية الشبكة بالشكل المبين سابقاً، حيث تحتوي على عصبوني دخل وعصبوني خرج وعصبونين في الطبقة الخفية. النقطة الهامة هي تحديد أوزان الترابطات بين العصبونات بحيث تحقق عملية المحاكاة المطلوبة. يمكن للأوزان أن تكون أي قيم عددية حقيقية.

الترميز:

يمكن حل المسألة بافتراض أن الصبغي عبارة عن سلسلة من الأعداد الحقيقية بطول يساوي عدد الترابطات أي الأوزان الموجودة في الشبكة المصممة، وهذه القيم الحقيقية تعبر عن أوزان الشبكة العصبونية . يفترض تحديد أرقام صحيحة متتالية للترابطات، وبحيث يكون دليل المصفوفة المعبرة عن الصبغي هو ترقيم الترابطات في الشبكة.

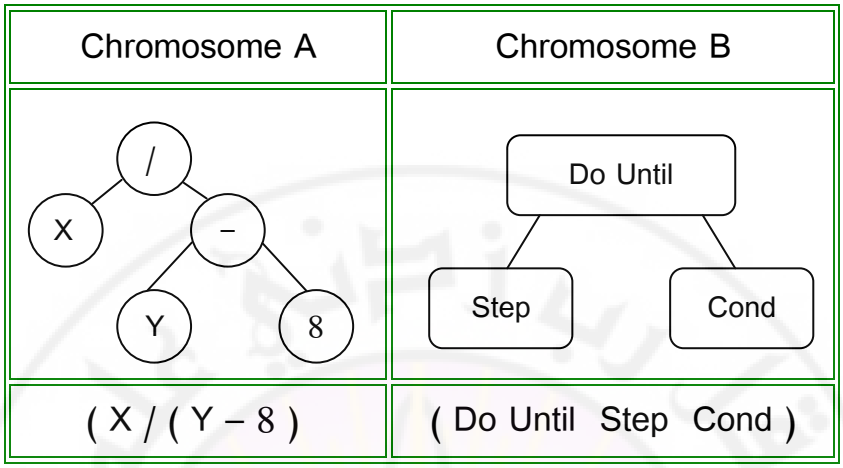
فإذا كان الصبغي بالشكل الآتي:

Chromosome A = **[2 −1 2 2 2 1 1]**

فهذا يعني أنه لدينا (7) ترابطات وبالتالي (7) أوزان. إن وزن خط الارتباط الأول هو (2)، ووزن خط الارتباط الثاني هو (1−)، ووزن خط الارتباط الثالث هو (2)، وهكذا لنهاية الصبغي. يحسب تابع الملاءمة باختبار الشبكة العصبونية، وكلما كانت أكثر دقة كان الصبغي أكثر قرباً من الحل الأمثل.

4. الترميز الشجري (Tree Encoding):

من أصعب أنواع الترميز حيث يستخدم بشكل رئيسي في برامج التطوير والخبرة كما في البرمجة الجينية (GP)، وفيها يكون كل صبغي عبارة عن شجرة مؤلفة من عدة عناصر كالتوابع والأوامر في اللغة البرمجية ، وهو مفيد في برامج التطوير أو في أي مشكلة ذات بنية شجرية أو يمكن تمثيلها ببنية شجرية. تستخدم لهذا الهدف لغة برمجية من صنف لغات الذكاء الصنعي مثل لغة (LISP)، حيث تكون البرامج فيها معطاة مباشرة بشكل شجري، ومن السهل أن ترمز بهذه الطريقة وبذلك ليس هناك صعوبة في عمليات (GA) مثل العبور والطفرة . يبين الشكل(6−12) بعض الصبغيات المرمزة شجرياً لمسألتين مختلفتين:



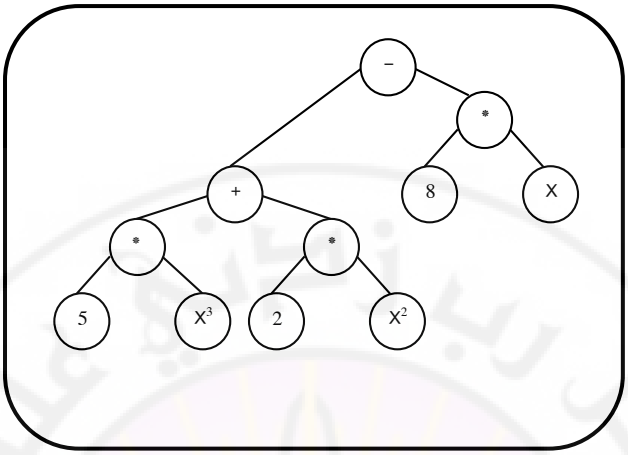
الشكل(6-12) ترميز الصبغيات شجرياً

مثال: إيجاد تابع انطلاقاً من عدة نقاط.

الترميز الشجري من أقل أنواع الترميز استخداماً نظراً لمتطلباته البرمجية المعقدة. في المثال البسيط الذي نورده يتبين لنا وجود العديد من النقاط الواجب مراعاتها عند استخدام الترميز الشجري.

يوضح الشكل(6-13) ترميزاً شجرياً لأحد التوابع الصحيحة من الدرجة الثالثة، والمعطى بالشكل:

$$f(x) = 5x^3 + 2x^2 - 8x$$



الشكل(6-13) ترميز شجري لأحد التوابع الصحيحة

وصف المسألة:

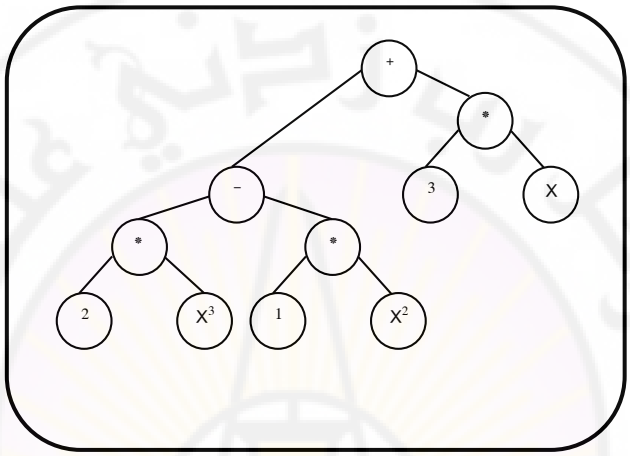
قام أحد المهندسين بتصميم دارة إلكترونية تمثيلية وحيدة الدخل ووحيدة الخرج، وبشكل تعطي قيمة جهد خرج وحيدة عند تطبيق جهد دخل ما. وبعد مجموعة من التجارب فقد حصل المصمم على الجدول أدناه. المطلوب، إيجاد التابع الصحيح الممثل لهذه النقاط:

جهد الدخل (V)	1	2	3	4	5	6	7
جهد الخرج (V)	4	18	54	124	240	414	658

الترميز:

يمكن حل المسألة بافتراض أن الصبغي عبارة عن شجرة مؤلفة من (11) عقدة بحيث تعطي ثلاثة حدود على الأكثر. دائماً تكون العقدة الأعلى تحتوي على

عملية رياضية بين العقدتين الفرعيتين. كما يمكن استخدام الأعداد الحقيقية، والعمليات الرياضية الأساسية الأربعة، والمتحول (x) حتى الدرجة الثالثة. يوضح الشكل(6-14) صبغياً يمثل حلاً مناسباً للمسألة المطروحة.



الشكل(6-14) الصبغي المرمز شجرياً والممثل حلاً للمسألة

إن التابع الصحيح وفق الصبغي يعطى بالشكل:

$$f(x) = 2x^3 - x^2 + 3x$$

6-11- مبادئ وأسس رياضية:

قبل تناول خوارزميات الانتخاب، لا بد من استعراض بعض المفاهيم الرياضية الاحتمالية، المستخدمة بشكل واسع في الخوارزميات الجينية، والتي تعتمد عليها خوارزميات الانتخاب خاصة.

1. احتمال الانتخاب (Probability of Selection (P_{select})):

هو احتمال انتخاب كل صبغي (فرد) من صبغيات (أفراد) الجيل الحالي لتوليد جيل الأبناء، ويعطى بالعلاقة:

$$P_{select}(i) = \frac{F_i}{\sum_{j=1}^{n} F_j}$$

حيث:

$P_{select}(i)$: احتمال انتخاب الصبغي i.

F_i: درجة ملاءمة الصبغي i.

n: عدد صبغيات الجيل (حجم الجيل).

2. القيمة المتوقعة لطلب الصبغي (Expected Number of Instances (m)):

تعبر هذه القيمة عن عدد مرات انتخاب الصبغي من مجموعة صبغيات الجيل الحالي:

$$m(i) = n \times P_{select}(i)$$

$$\Rightarrow m(i) = n \times \frac{F_i}{\sum_{j=1}^{n} F_j}$$

$$\Rightarrow m(i) = \frac{F_i}{\overline{F}}$$

حيث:

$m(i)$: القيمة المتوقعة لطلب الصبغي i.

$\overline{F}$: متوسط درجة الملاءمة لصبغيات الجيل.

3. ضغط الانتخاب ((Selection pressure (S)):

هو حاصل قسمة متوسط درجة الملاءمة لصبغيات الجيل الحالي على متوسط درجة الملاءمة لصبغيات الجيل السابق (جيل االسلف):

$$s = \frac{\overline{F_1}}{\overline{F_o}}$$

مثال:

عند حل إحدى المسائل باستخدام الخوارزميات الجينية توصلنا إلى المعطيات المبينة في الجدول أدناه:

الجيل الثالث						
الصبغي	1	2	3	4	5	6
درجة الملاءمة	5	9	13	10	1	4
الجيل الرابع						
الصبغي	1	2	3	4	5	6
درجة الملاءمة	6	7	13	15	4	3

يمكن حساب احتمال انتخاب كل فرد من أفراد الجيل الثالث بالشكل:

$$P_{select}\,(i) = \frac{F_i}{\sum_{j=1}^{n} F_j} \Rightarrow$$

$$P_{select}\,(1) = \frac{F_1}{\sum_{j=1}^{6} F_j} = \frac{5}{5+9+13+10+1+4} = \frac{5}{42} \approx 0.119$$

$$P_{select}\,(2) = \frac{F_2}{\sum_{j=1}^{6} F_j} = \frac{9}{5+9+13+10+1+4} = \frac{9}{42} \approx 0.214$$

$$P_{select}\,(3) = \frac{F_3}{\sum_{j=1}^{6} F_j} = \frac{13}{5+9+13+10+1+4} = \frac{13}{42} \approx 0.310$$

$$P_{select}\,(4) = \frac{F_4}{\sum_{j=1}^{6} F_j} = \frac{10}{5+9+13+10+1+4} = \frac{10}{42} \approx 0.238$$

$$P_{select}\,(5) = \frac{F_5}{\sum_{j=1}^{6} F_j} = \frac{1}{5+9+13+10+1+4} = \frac{1}{42} \approx 0.024$$

$$P_{select}\,(6) = \frac{F_6}{\sum_{j=1}^{6} F_j} = \frac{4}{5+9+13+10+1+4} = \frac{4}{42} \approx 0.095$$

$$\sum_{i=1}^{6} P_{select}\,(i) = 0.119 + 0.214 + 0.310 + 0.238 + 0.024 + 0.095 = 1$$

نلاحظ أن أعلى درجة ملاءمة في الجيل الثالث هي للصبغي الثالث والتي تساوي إلى (13)، وبالتالي فإن احتمال انتخاب هذا الصبغي أو الفرد هو الأعلى بين بقية أفراد الجيل الثالث والذي يساوي (0.310). أما أدنى درجة ملاءمة في الجيل الثالث فهي للصبغي الخامس والتي تساوي إلى (1)، وبالتالي فإن احتمال

انتخاب هذا الصبغي أو الفرد هو الأدنى بين بقية أفراد الجيل الثالث والذي يساوي (0.024). كما نلاحظ أن مجموع الاحتمالات يساوي الواحد.

كما يمكن حساب القيمة المتوقعة لطلب الصبغي في الجيل الثالث بالشكل:

$$m(i) = n \times P_{select}\,(i) \Rightarrow$$

$$m(1) = 6 \times P_{select}\,(1) = 6 \times 0.119 = 0.714$$

$$m(2) = 6 \times P_{select}\,(2) = 6 \times 0.214 = 1.284$$

$$m(3) = 6 \times P_{select}\,(3) = 6 \times 0.310 = 1.860$$

$$m(4) = 6 \times P_{select}\,(4) = 6 \times 0.238 = 1.428$$

$$m(5) = 6 \times P_{select}\,(5) = 6 \times 0.024 = 0.144$$

$$m(6) = 6 \times P_{select}\,(6) = 6 \times 0.095 = 0.570$$

نلاحظ أن كل فرد ذي درجة ملاءمة أعلى من المتوسط الحسابي لدرجات ملاءمة الجيل، سوف تكون القيمة المتوقعة لطلبه أكبر من الواحد. بينما كل فرد ذي درجة ملاءمة أدنى من المتوسط الحسابي لدرجات ملاءمة الجيل، سوف تكون القيمة المتوقعة لطلبه أصغر من الواحد.

كما يمكن حساب ضغط الانتخاب بين الجيل الثالث والجيل الرابع بالشكل:

$$S = \frac{\overline{F_1}}{\overline{F_0}} \Rightarrow$$

$$S = \frac{\overline{F_4}}{\overline{F_3}} \Rightarrow$$

$$S = \frac{\frac{6+7+13+15+4+3}{6}}{\frac{5+9+13+10+1+4}{6}} = \frac{8}{7} \approx 1.143$$

نلاحظ أن ضغط الانتخاب أكبر من الواحد، وبالتالي فإن الجيل الرابع أفضل من الجيل الثالث، وهناك اقتراب تدريجي من الحل الأمثل.

6-12- خوارزميات الانتخاب (Selection):

يعد الانتخاب أول مرحلة من مراحل الخوارزميات الجينية الثلاثة، وهي الانتخاب والعبور والطفرة. تنتخب الآباء في هذه المرحلة من أفراد الجيل الحالي لتتم عليهم باقي إجرائيات الخوارزميات الجينية، أي يتم اختيار الأفراد المناسبين من هذا الجيل حسب طريقة الملاءمة المستخدمة، وبطريقة موضوعية وغير متحيزة لتوليد أفراد الجيل التالي (توليد أبناء جديدة) . إن وجود أكثر من خوارزمية انتخاب يعني وجوب قيام المستخدم باختيار خوارزمية الانتخاب المناسبة لمسألته في مرحلة التأهيل الابتدائي من المخطط التدفقي للخوارزمية الجينية، علماً أن أداء خوارزمية الانتخاب يتعلق بنوع المسألة المطروحة، وهذا ما أدى لوجود دراسات عديدة لمقارنة أداء خوازميات الانتخاب في حل بعض المسائل المشهورة، فالقرار الخاص باستخدام خوازمية انتخاب ما يخضع لدراسة المسألة والظروف المحيطة

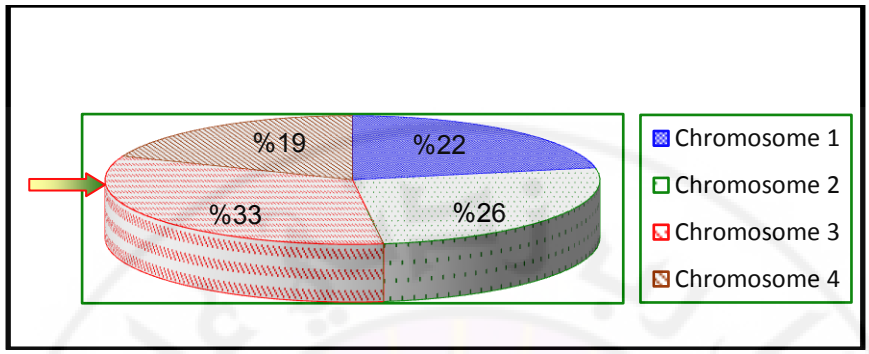
بها، فهو من القرارات الهامة التي يعتمد عليها سير الخوارزمية وسرعة الوصول إلى الحل الأمثل.

من المناسب هنا، الإشارة إلى أنه عندما نذكر جملة، " قيام المستخدم باختيار خوارزمية الانتخاب المناسبة "، أو ما شابهها من العبارات فإننا نتحدث عن نظام برمجي متكامل لحل المسائل باستخدام الخوارزميات الجينية. حيث يتوافر فيه مجموعة واسعة من الخيارات التي يحددها المستخدم، وكلما قام المستخدم باختيار ما، تظهر لمجموعة جديدة من الخيارات المتعلقة بخياره الأول. فعندما يختار خوارزمية انتخاب ما، تظهر له المحددات المتعلقة بهذه الخوارزمية، ليختار ما يناسب مسألته، أو إجراء عمليات المقارنة بهدف بحثي.

سوف نتناول في الفقرات اللاحقة أهم خوارزميات الانتخاب المستخدمة في حل المسائل المعقدة.

1. خوارزمية الانتخاب على مبدأ العجلة المتدحرجة (Roulette Wheel Selection):

تعد من أشهر خوارزميات الانتخاب في الخوارزميات الجينية وأكثرها استخداماً سواء منفردة أم كمرحلة تالية لخوارزمية انتخاب أخرى. في هذه الطريقة ، والتي تسمى أحياناً بخوارزمية الانتخاب التناسبي (Proportional selection)، توزع صبغيات الجيل الحالي على عدة قطاعات لعجلة مقسمة إلى (100) قطاع، حسب احتمال الانتخاب (P_{select}) لكل صبغي ، يوضح الشكل(6-15) تمثيلاً للعجلة المتدحرجة الممثلة لجيل مؤلف من أربعة صبغيات. يمثل الصبغي ذي درجة الملاءمة الأكبر بمساحة قطاعية أكبر في العجلة. فإذا كان احتمال انتخاب الصبغي الثالث يساوي (0.333) فرضاً، فهذا يعني تخصيص ثلث مساحة العجلة لهذا الصبغي.



الشكل(6-15) الانتخاب على مبدأ العجلة المتدحرجة

فإذا أديرت العجلة بشكل عشوائي حتى وقوفها عند مؤشر ثابت، عندها ينتخب الصبغي التابع للقطاع الذي أشار إليه المؤشر. مما لا شك فيه، أن احتمال وقوف المؤشر عند المساحة الأكبر، أعلى من احتمال وقوفه عند المساحة الأصغر. لمحاكاة عملية العجلة ودورانها حاسوبياً، فإن كل صبغي يعطى قيمة تعبر عن الاحتمال التراكمي له (C)، وتحسب بالعلاقة:

$$C(i) = \sum_{j=1}^{i} F_j \Rightarrow$$

$$C(1) = F_1$$
$$C(2) = F_1 + F_2$$
$$C(3) = F_1 + F_2 + F_3$$
$$C(4) = F_1 + F_2 + F_3 + F_4$$

توزع قطاعات العجلة على الصبغيات حسب الاحتمال التراكمي المحسوب بالعلاقة السابقة، ثم يولد رقماً عشوائياً حاسوبياً ضمن المجال [0,100]، لينتخب الصبغي التابع للقطاع الذي وقع رقمه على الرقم العشوائي المولد ، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ستة صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5	6
درجة الملاءمة	3	15	2	10	6	4

نحسب احتمال انتخاب الصبغيات لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
احتمال الانتخاب	0.075	0.375	0.050	0.250	0.150	0.100

وبالتالي فإن الاحتمال التراكمي للصبغيات يعطى وفق الجدول الآتي:

الصبغي	1	2	3	4	5	6
الاحتمال التراكمي	0.075	0.450	0.500	0.750	0.900	1.000

لمحاكاة عملية العجلة المتدحرجة حاسوبياً فإننا نقوم بضرب احتمال الانتخاب التراكمي بمائة لنحصل على القيم الآتية:

الصبغي	1	2	3	4	5	6
القيمة الجديدة	7.5	45.0	50.0	75.0	90.0	100.0

بعد ذلك نولد تتابعياً بحجم الجيل أعداداً عشوائية ضمن المجال [0,100]، وينتخب الصبغي الذي يقع العدد العشوائي ضمن مجاله، كما هو مبين في الجدول الآتي:

العدد العشوائي	23.0	57.1	98.4	86.3	32.6	64.1
الصبغي المنتخب	2	4	6	5	2	4

نلاحظ أنه عند توليد العدد العشوائي فإننا ننتخب أدنى صبغي ذي الاحتمال التراكمي الجديد الأعلى أو يساوي العدد العشوائي المولد. فعند توليد العدد (23.0) نختار الصبغي الثاني لأن الاحتمال التراكمي الجديد له (الاحتمال التراكمي مضروباً بمائة) يساوي (45.0)، وهو أدنى الصبغي الأدنى في تحقيق أن احتماله التراكمي الجديد أعلى أو يساوي العدد (23.0)، وهكذا لبقية الصبغيات. من الممكن البدء بطرح القيمة الجديدة في الجدول من العدد العشوائي المولد بدءاً من الصبغي الأول وحتى الحصول على قيمة أصغر من الصفر أو تساويه ليتم انتخاب هذا الصبغي.

كما أنه من الملفت للانتباه تكرار انتخاب الصبغي الثاني والرابع، وهذا منطقي، فهما يملكان درجة الملاءمة الأكبر، وبالتالي فاحتمال انتخابهم أعلى، وسنرى لاحقاً التأثير الإيجابي لهذه العملية التي تؤدي إلى نقل مورثات (جينات)

الآباء الأعلى درجة ملاءمة إلى الأبناء في الجيل الخلف، ما يؤثر على توجيه الحلول نحو الحل الأمثل أو الأفضل في فضاء الحل.

وبالسياق نفسه، نقول أن الصبغي الأول والثالث لم ينتخبا، وهذا أيضاً منطقي، فهما الأدنى درجة ملاءمة في الجيل الحالي وانتخابهما سوف يؤدي إلى نقل مورثات سيئة من جيل السلف إلى جيل الخلف.

تبقى هذه العمليات احتمالية، فمن الممكن أن تلعب الصدفة المحضة دوراً سلبياً، كأن تولد أعداداً عشوائية تؤدي لانتخاب الصبغي الثالث مثلاً عدة مرات، مع عدم انتخاب الصبغي الثاني مثلاً.

2. خوارزمية الانتخاب التسلسلي الباقي مع الاستبدال (Stochastic Remainder With Replacement Selection):

في هذه الخوارزمية يتم حساب القيمة المتوقعة لطلب الصبغي (m)، لكل صبغي من صبغيات الجيل، ثم يؤخذ القسم الصحيح من هذا العدد. إذا كان القسم الصحيح أكبر من الواحد أو يساويه يتم انتخاب هذا الصبغي، أما إذا كان يساوي الصفر فيتم رفض هذا الصبغي، وهذا يعني انتخاب الصبغيات التي لديها درجة ملاءمة أعلى أو تساوي المتوسط الحسابي لدرجات ملاءمة صبغيات الجيل . لتكملة النقص في عدد الصبغيات اللازمة للانتخاب، يتم أخذ القسم الكسري للقيمة المتوقعة لطلب الصبغي واعتباره مقياس ملاءمة جديد له، ومن ثم يتم تطبيق الطريقة السابقة للانتخاب على مبدأ العجلة المتدحرجة لانتخاب بقية الصبغيات، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ستة صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5	6
درجة الملاءمة	3	15	2	10	6	4

نحسب احتمال انتخاب الصبغيات لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
احتمال الانتخاب	0.075	0.375	0.050	0.250	0.150	0.100

نحسب القيمة المتوقعة لطلب الصبغي (m)، كما نأخذ القسم الصحيح والقسم الكسري لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
القيمة (m)	0.450	2.250	0.300	1.500	0.900	0.600
القسم الصحيح	0	2	0	1	0	0
القسم الكسري	0.450	0.250	0.300	0.500	0.900	0.600

الآن، ينتخب الصبغي ذي القسم الصحيح الأكبر من الواحد أو يساويه، وبالتالي ينتخب الصبغي الثاني والرابع. يجب انتخاب عدد من الصبغيات يساوي حجم الجيل لذلك يستكمل النقص بانتخاب أربعة صبغيات بأخذ القسم الكسري للقيمة المتوقعة لطلب الصبغي واعتباره مقياس ملاءمة جديد له، ومن ثم تطبق

الطريقة السابقة للانتخاب على مبدأ العجلة المتدحرجة لانتخاب أربعة صبغيات فقط، وذلك بالشكل الآتي:

الصبغي	1	2	3	4	5	6
الملاءمة الجديدة	0.450	0.250	0.300	0.500	0.900	0.600

نحسب الاحتمالات الجديدة لانتخاب الصبغيات لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
احتمال الانتخاب	0.150	0.083	0.100	0.167	0.300	0.200

وبالتالي فإن الاحتمال التراكمي الجديد للصبغيات، يعطى وفق الجدول الآتي:

الصبغي	1	2	3	4	5	6
الاحتمال التراكمي	0.150	0.233	0.333	0.500	0.800	1.000

لمحاكاة عملية العجلة المتدحرجة حاسوبياً، فإننا نقوم بضرب احتمال الانتخاب التراكمي بمائة لنحصل على القيم الآتية:

الصبغي	1	2	3	4	5	6
القيمة الجديدة	15.0	23.3	33.3	50.0	80.0	100.0

بعد ذلك، نولد تتابعياً من الأعداداً العشوائية ضمن المجال [0,100]، مع الانتباه إلى أن عدد الأعداد العشوائية يساوي إلى حجم الجيل مطروحاً منه عدد

الصبغيات ذات القسم الصحيح الأكبر من الواحد أو يساويه وينتخب الصبغي الذي يقع العدد العشوائي ضمن مجاله، كما هو مبين في الجدول الآتي:

العدد العشوائي	41.5	67.2	88.9	16.1
الصبغي المنتخب	4	5	6	2

وهكذا يصبح عدد الصبغيات المنتخبة يساوي الستة بحجم الجيل، وهي الصبغيات المبينة في الجدول الآتي:

الصبغي المنتخب	2	4	4	5	6	2

الجزء الأول من الصبغيات المنتخبة ينتج عن القسم الصحيح، والجزء الثاني ينتج عن خوارزمية العجلة المتدحرجة للقسم الكسري.

3. خوارزمية الانتخاب التسلسلي الباقي بدون الاستبدال (Stochastic Remainder Without Replacement Selection):

هذه الخوارزمية مماثلة لخوارزمية الانتخاب التسلسلي الباقي مع الاستبدال، سوى أن الاختلاف الوحيد هو في مرحلة تكملة عدد الصبغيات اللازمة للانتخاب، فهنا يتم أيضاً استخدام مبدأ العجلة المتدحرجة ولكن بعد حذف الصبغي المنتخب في كل مرة انتخاب، ويعاد حساب احتمالية الانتخاب للصبغيات المتبقية من جديد، وإعادة تطبيق مبدأ العجلة المتدحرجة من جديد حتى يستكمل النقص في حجم الجيل، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ستة صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5	6
درجة الملاءمة	3	15	2	10	6	4

نحسب احتمال انتخاب الصبغيات لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
احتمال الانتخاب	0.075	0.375	0.050	0.250	0.150	0.100

نحسب القيمة المتوقعة لطلب الصبغي (m)، كما نأخذ القسم الصحيح والقسم الكسري لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
القيمة (m)	0.450	2.250	0.300	1.500	0.900	0.600
القسم الصحيح	0	2	0	1	0	0
القسم الكسري	0.450	0.250	0.300	0.500	0.900	0.600

الآن، ينتخب الصبغي ذي القسم الصحيح الأكبر من الواحد أو يساويه، وبالتالي ينتخب الصبغي الثاني والرابع. يجب انتخاب عدد من الصبغيات يساوي حجم الجيل لذلك يستكمل النقص بانتخاب أربعة صبغيات بأخذ القسم الكسري للقيمة المتوقعة لطلب الصبغي واعتباره مقياس ملاءمة جديد له، ومن ثم تطبق الطريقة السابقة للانتخاب على مبدأ العجلة المتدحرجة لانتخاب أربعة صبغيات فقط، وذلك وفق الجدول الآتي:

الصبغي	1	2	3	4	5	6
الملاءمة الجديدة	0.450	0.250	0.300	0.500	0.900	0.600

نحسب الاحتمالات الجديدة لانتخاب الصبغيات لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
احتمال الانتخاب	0.150	0.083	0.100	0.167	0.300	0.200

وبالتالي فإن الاحتمال التراكمي الجديد للصبغيات يعطى وفق الجدول الآتي:

الصبغي	1	2	3	4	5	6
الاحتمال التراكمي	0.150	0.233	0.333	0.500	0.800	1.000

لمحاكاة عملية العجلة المتدحرجة حاسوبياً فإننا نقوم بضرب احتمال الانتخاب التراكمي بمائة لنحصل على القيم الآتية:

الصبغي	1	2	3	4	5	6
القيمة الجديدة	15.0	23.3	33.3	50.0	80.0	100.0

بعد ذلك، نولد عدداً عشوائياً واحداً ضمن المجال [0,100]، وينتخب الصبغي الذي يقع العدد العشوائي ضمن مجاله، كما هو مبين في الجدول الآتي:

العدد العشوائي	69.3
الصبغي المنتخب	5

وهكذا تصبح الصبغيات المنتخبة هي الصبغي الثاني والرابع والخامس.

الآن، نحذف الصبغي الخامس ليصبح الجدول الجديد كما يأتي:

الصبغي	1	2	3	4		6
الملاءمة الجديدة	0.450	0.250	0.300	0.500		0.600

نحسب الاحتمالات الجديدة لانتخاب الصبغيات على اعتبار عدد الصبغيات الجديد هو خمسة صبغيات فقط لنحصل على الجدول الآتي:

الصبغي	1	2	3	4		6
احتمال الانتخاب	0.214	0.119	0.143	0.238		0.286

وبالتالي فإن الاحتمال التراكمي الجديد للصبغيات يعطى وفق الجدول الآتي:

الصبغي	1	2	3	4		6
الاحتمال التراكمي	0.214	0.333	0.476	0.714		1.000

بضرب احتمال الانتخاب بمائة نحصل على القيم الآتية:

الصبغي	1	2	3	4		6
القيمة الجديدة	21.4	33.3	47.6	71.4		100.0

بعد ذلك، نولد عدداً عشوائياً واحداً ضمن المجال [0,100]، وينتخب الصبغي الذي يقع العدد العشوائي ضمن مجاله، كما هو مبين في الجدول الآتي:

العدد العشوائي	18.2
الصبغي المنتخب	1

وهكذا تصبح الصبغيات المنتخبة هي الصبغي الثاني والرابع والخامس والأول.

الآن، نحذف الصبغي الأول ليصبح الجدول الجديد كما يأتي:

الصبغي		2	3	4		6
الملاءمة الجديدة		0.250	0.300	0.500		0.600

نحسب الاحتمالات الجديدة لانتخاب الصبغيات على اعتبار عدد الصبغيات الجديد هو أربعة صبغيات فقط لنحصل على الجدول الآتي:

الصبغي		2	3	4		6
احتمال الانتخاب		0.152	0.182	0.303		0.364

وبالتالي فإن الاحتمال التراكمي الجديد للصبغيات يعطى وفق الجدول الآتي:

الصبغي		2	3	4		6
الاحتمال التراكمي		0.152	0.182	0.637		1.000

مع الانتباه إلى تقريب الاحتمال التراكمي للصبغي السادس من القيمة (1.001) إلى القيمة (1.000) وذلك بسبب التقريب لثلاث خانات بعد الفاصلة ما يؤدي لحدوث بعض الأخطاء التقريبية التي نحلها بوجوب كون الاحتمال التراكمي للصبغي الأخير يساوي الواحد حصراً.

بضرب احتمال الانتخاب التراكمي بمائة نحصل على القيم الآتية:

الصبغي		2	3	4		6
القيمة الجديدة		15.2	18.2	63.7		100.0

بعد ذلك، نولد عدداً عشوائياً واحداً ضمن المجال [0,100]، وينتخب الصبغي الذي يقع العدد العشوائي ضمن مجاله، كما هو مبين في الجدول الآتي:

العدد العشوائي	28.2
الصبغي المنتخب	4

وهكذا تصبح الصبغيات المنتخبة هي الصبغي الثاني والرابع والخامس والأول والرابع أيضاً.

الآن، نحذف الصبغي الرابع ليصبح الجدول الجديد كما يأتي:

الصبغي		2	3			6
الملاءمة الجديدة		0.250	0.300			0.600

نحسب الاحتمالات الجديدة لانتخاب الصبغيات على اعتبار عدد الصبغيات الجديد هو ثلاثة صبغيات فقط لنحصل على الجدول الآتي:

الصبغي		2	3			6
احتمال الانتخاب		0.217	0.261			0.522

وبالتالي فإن الاحتمال التراكمي الجديد للصبغيات يعطى وفق الجدول الآتي:

الصبغي		2	3			6
الاحتمال التراكمي		0.217	0.478			1.000

بضرب احتمال الانتخاب التراكمي بمائة نحصل على القيم الآتية:

الصبغي		2	3			6
القيمة الجديدة		21.7	47.8			100.0

بعد ذلك، نولد عدداً عشوائياً واحداً ضمن المجال [0,100]، وينتخب الصبغي الذي يقع العدد العشوائي ضمن مجاله، كما هو مبين في الجدول الآتي:

العدد العشوائي	71.2
الصبغي المنتخب	6

وهكذا تصبح الصبغيات المنتخبة هي الصبغي الثاني والرابع والخامس والأول والرابع أيضاً والسادس. بما أن عدد الصبغيات المنتخبة أصبح يساوي حجم الجيل، فإننا نتوقف عن تكرار العمليات السابقة في خوارزمية الانتخاب التسلسلي الباقي بدون الاستبدال.

4. خوارزمية الانتخاب الإجباري (Deterministic Select):

هذه الخوارزمية مماثلة أيضاً لخوارزمية الانتخاب التسلسلي الباقي مع الاستبدال، سوى أن الاختلاف الوحيد هو في مرحلة تكملة عدد الصبغيات اللازمة للانتخاب، فهنا يتم ترتيب الصبغيات من جديد بشكل تنازلي على حسب القسم الحقيقي للقيمة المتوقعة لطلب الصبغي (m)، ومن ثم يؤخذ أول صبغي ثم الذي يليه حتى الوصول للعدد المطلوب للإكمال. ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ستة صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5	6
درجة الملاءمة	3	15	2	10	6	4

نحسب احتمال انتخاب الصبغيات لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
احتمال الانتخاب	0.075	0.375	0.050	0.250	0.150	0.100

نحسب القيمة المتوقعة لطلب الصبغي (m)، كما نأخذ القسم الصحيح والقسم الكسري لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6
القيمة (m)	0.450	2.250	0.300	1.500	0.900	0.600
القسم الصحيح	0	2	0	1	0	0
القسم الكسري	0.450	0.250	0.300	0.500	0.900	0.600

الآن، ينتخب الصبغي ذي القسم الصحيح الأكبر من الواحد أو يساويه، وبالتالي ينتخب الصبغي الثاني والرابع. يجب انتخاب عدد من الصبغيات يساوي حجم الجيل لذلك يستكمل النقص بترتيب الصبغيات من جديد بشكل تنازلي على حسب القسم الحقيقي للقيمة المتوقعة لطلب الصبغي (m)، ومن ثم يؤخذ أول

صبغي ثم الذي يليه حتى الوصول للعدد المطلوب للإكمال ، وذلك وفق الجدول الآتي:

الصبغي المرتب	5	6	4	1	3	2
القسم الكسري	0.900	0.600	0.500	0.450	0.300	0.250

بما أننا انتخبنا صبغيين فإننا بحاجة إلى انتخاب أربعة صبغيات أخرى حتى يصبح عدد الصبغيات المنتخبة يساوي حجم الجيل. نختار الصبغي الخامس والسادس والرابع والأول حسب تناقص القسم الكسري للقيمة المتوقعة لطلب الصبغي (m)، لتصبح الصبغيات المنتخبة كاملة هي الصبغي الأول والثاني والرابع (مرتين) والخامس والسادس.

5. خوارزمية الانتخاب على مبدأ مسابقة (n) عضو (n–Member Tournament Selection):

ينتخب بشكل عشوائي عدد من الصبغيات أو الأعضاء يساوي (n) صبغي من الجيل الحالي، ثم تجرى مسابقة بين الصبغيات على حسب مقياس الملاءمة لها، لينتخب الصبغي الذي يملك مقياس ملاءمة أعلى من غيره . تكرر العملية السابقة حتى يستكمل الجيل، أي عدداً من المرات يساوي حجم الجيل . تحدد قيمة (n) في مرحلة التأهيل الابتدائي من قبل المستخدم، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ستة صبغيات لها قيم الملاءمة المبينة في الجدول الآتي، كما أن قيمة (n) المدخلة في مرحلة التأهيل الابتدائي من قبل المستخدم هي (3):

الصبغي	1	2	3	4	5	6
درجة الملاءمة	3	15	2	10	6	4

يولد ثلاثة أعداد عشوائية صحيحة مختلفة ضمن المجال المعطى بالصيغة العامة [1,population size]=[1,6] لنحصل على الجدول الآتي:

العدد العشوائي	3	5	1
الصبغي المنتخب للمسابقة	3	5	1
درجة الملاءمة	2	6	3

ينتخب الصبغي ذي درجة الملاءمة الأفضل وهنا تكون القيمة الأعلى، أي الصبغي الخامس.

نكرر العملية السابقة بتوليد ثلاثة أعداد عشوائية صحيحة ضمن المجال [1,6] لنحصل على الجدول الآتي:

العدد العشوائي	2	1	6
الصبغي المنتخب للمسابقة	2	1	6
درجة الملاءمة	15	3	4

ينتخب الصبغي ذي درجة الملاءمة الأعلى، أي الصبغي الثاني، ليصبح لدينا صبغيين منتخبين هما الصبغي الخامس والثاني.

تكرر العملية السابقة أربع مرات لنحصل على الجدول الآتي:

العدد العشوائي	3	2	6
الصبغي المنتخب للمسابقة	3	2	6
درجة الملاءمة	2	15	4
الصبغي المنتخب: الثاني			
العدد العشوائي	4	5	2
الصبغي المنتخب للمسابقة	4	5	2
درجة الملاءمة	10	6	15
الصبغي المنتخب: الثاني			
العدد العشوائي	4	3	6
الصبغي المنتخب للمسابقة	4	3	6
درجة الملاءمة	10	2	4
الصبغي المنتخب: الرابع			
العدد العشوائي	1	6	4
الصبغي المنتخب للمسابقة	1	6	4
درجة الملاءمة	3	4	10
الصبغي المنتخب: الرابع			

أصبح عدد الصبغيات المنتخبة يساوي الستة، أي يساوي حجم الجيل وهي الصبغي الخامس والثاني والثاني والثاني والرابع والرابع.

6. خوارزمية المرتبة (Rank Sale):

تظهر لخوارزمية العجلة المتدحرجة عدة مساوئ عندما يكون هناك تفاوت كبير في قيم مقياس الملاءمة لصبغيات الجيل. فمثلاً عندما يكون مقياس الملاءمة لصبغي (90%) من مجموع قيم مقياس الملاءمة لك افة صبغيات الجيل، فعندها سوف تملك بقية الصبغيات فرص اختيار قليلة جداً.

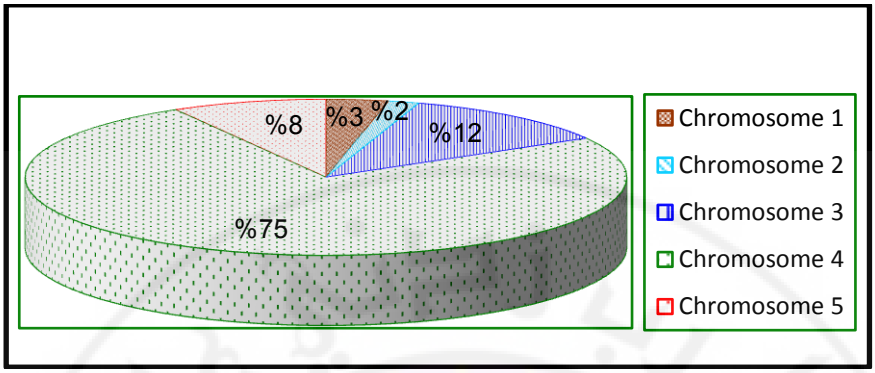
في خوارزمية المرتبة تعطى مرتبة لصبغيات الجيل حسب الترتيب التصاعدي لقيم تابع مقياس الملاءمة، فيعطى الصبغي ذي قيمة تابع الملاءمة الأسوأ المرتبة واحد، والذي أفضل منه يعطى المرتبة اثنين، والأفضل يعطى مرتبة ثلاث، وهكذا. يكون أفضل صبغي ذي مرتبة (n) وهي حجم الجيل، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من خمسة صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5
درجة الملاءمة	2	1	7	45	5

يبين الشكل(6-16) توزع الصبغيات على العجلة المتدحرجة وفق قطاعات تتناسب مساحتها مع درجة الملاءمة لكل صبغي:



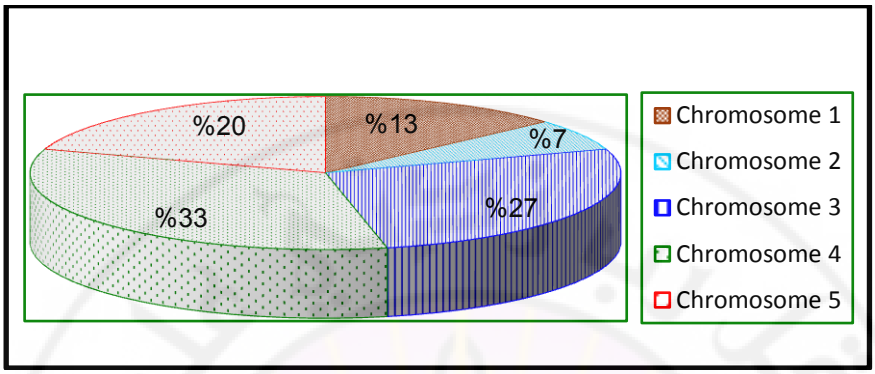
الشكل(6-16) توزع الصبغيات حسب خوارزمية العجلة المتدحرجة

في حال محاكاة العجلة المتدحرجة حاسوبياً فإن فرصة اختيار وتكرار الصبغي الرابع كبيرة، وتساوي ثلاثة أرباع فرصة أو احتمال اختيار بقية الصبغيات في الجيل. بمعنى آخر، ووفق نظرية الاحتمالات فمن الممكن وسطياً انتخاب الصبغي الرابع (3.75) مرة، أي حوالي أربع تكرارات للصبغي نفسه، مع صبغي واحد من الصبغيات الأربعة الأخرى.

ولحل هذه المشكلة، التي تظهر خاصة عندما تكون هناك فروق كبيرة في الملاءمة بين صبغيات الجيل الواحد ، تستخدم طريقة المرتبة التي تجعل جميع الصبغيات تمتلك فرصاً في الانتخاب بترتيبها تصاعدياً كما هو مبين في الجدول:

الصبغي	2	1	5	3	4
درجة الملاءمة	1	2	5	7	45
المرتبة	1	2	3	4	5

يبين الشكل(6-17) التوزيع الجديد للصبغيات على العجلة المتدحرجة باعتبار درجة الملاءمة هي المرتبة.



الشكل(6-17) الانتخاب على طريقة المرتبة

يبين الجدول الآتي النسبة المئوية التقريبية لمساحة القطاع لكل صبغي وفق طريق المرتبة:

الصبغي	2	1	5	3	4
المرتبة	1	2	3	4	5
المرتبة	7%	13%	20%	27%	33%

من سيئات هذه الطريقة أن ها تقود لتقارب في احتمالات انتخاب الصبغيات، حيث لا تختلف الصبغيات الأفضل عن بقية الصبغيات كثيراً، فبعد أن كانت درجة ملاءمة الصبغي الرابع ثلاثة أرباع مجموع درجات الملاءمة لكامل صبغيات الجيل، أصبحت ثلث مجموع درجات الملاءمة لكامل صبغيات الجيل.

7. خوارزمية الحالة الثابتة (Steady State Selection):

تهدف خوارزمية الحالة الثابتة إلى التخلص من الصبغيات ذات درجة الملاءمة السيئة، وانتخاب الصبغيات ذات درجة الملاءمة الجيدة فقط. تنفذ هذه الخوارزمية بانتخاب الصبغيات الجيدة أولاً، ومن ثم انتخاب بقية الصبغيات من

مجموعة الصبغيات الجيدة التي انتخبت أولاً. فيكون مجموعة الصبغيات المنتخبة عبارة عن قسمين، القسم الأول هو صبغيات الجيل الجيدة نفسها، والقسم الثاني هو الصبغيات الناتجة من الصبغيات الجيدة . إن عدد الصبغيات المنتخبة في القسم الأول يحدده المستخدم في مرحلة التأهيل الابتدائي، إما كقيمة مطلقة أصغر من حجم الجيل، أو كنسبة مئوية من حجم الجيل. كما أن طريقة انتخاب القسم الثاني هي من خيارات المستخدم، التي يحددها في نظامه البرمجي عند اختياره خوارزمية انتخاب الحالة الثابتة، فمن الممكن اختيار خوازمية العجلة المتدحرجة أو غيرها من خوارزميات الانتخاب، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ثمانية صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5	6	7	8
درجة الملاءمة	11	10	8	11	10	20	15	5

نفترض كما الأمثلة السابقة، أن الصبغي ذي درجة الملاءمة الأكبر هو الأفضل. نجد من الجدول أن الصبغي الأول والرابع لهما درجة الملاءمة نفسها، وهذا أمر عادي، فلا يوجد ما يمنع تساوي درجة الملاءمة بين عدة صبغيات مختلفة في مورثاتها.

ينتخب القسم الأول بانتخاب عدد من الصبغيات الجيدة، وليكن العدد المدخل من قبل المستخدم في مرحلة التأهيل الابتدائي هو قيمة مطلقة أصغر من حجم الجيل وتساوي الأربعة، لذلك ينتخب أفضل أربعة صبغيات من الجيل الحالي، وهذا ما يتضمنه الجدول الآتي:

القسم الأول من الصبغيات المنتخبة	1	4	6	7
درجة الملاءمة	11	11	20	15

لانتخاب القسم الثاني فإننا نعتبر جيلاً جديداً هو القسم الأول، لننتخب منه عدداً من الصبغيات يكمل النقص في حجم الجيل، والذي يحسب بطرح عدد صبغيات القسم الأول من حجم الجيل، وبالتالي أربعة صبغيات. لتكن خوارزمية الانتخاب المحددة من قبل المستخدم في مرحلة التأهيل الابتدائي كخوارزمية انتخاب للقسم الثاني في خوارزمية الحالة الثابتة هي خوارزمية الانتخاب على مبدأ مسابقة (n) عضو. هذا يفرض أيضاً قيام المستخدم بتحديد عدد أعضاء المسابقة وليكن ثلاثة أعضاء.

كما بينا في مثال سابق فإنه يولد ثلاثة أعداد عشوائية صحيحة مختلفة ضمن المجال [1,4]، وينتخب الصبغي الأفضل. مع الانتباه إلى أن ترقيم الصبغيات أصبح غير متتابع، لذلك يؤخذ دليل المصفوفة المتضمنة صبغيات القسم الأول، وتكرر العملية أربع مرات لنحصل على الجدول الآتي:

العدد العشوائي	1	2	4
الصبغي المنتخب للمسابقة	1	4	7
درجة الملاءمة	11	11	15
الصبغي المنتخب : السابع			
العدد العشوائي	2	4	3
الصبغي المنتخب للمسابقة	4	7	6
درجة الملاءمة	11	15	20

الصبغي المنتخب : السادس			
العدد العشوائي	2	4	3
الصبغي المنتخب للمسابقة	4	7	6
درجة الملاءمة	11	15	20
الصبغي المنتخب : السادس			
العدد العشوائي	4	1	2
الصبغي المنتخب للمسابقة	7	1	4
درجة الملاءمة	15	11	11
الصبغي المنتخب : السابع			

أصبح عدد الصبغيات المنتخبة يساوي الثمانية، أي يساوي حجم الجيل وهي المبينة في الجدول الآتي:

القسم الأول من الصبغيات المنتخبة	1	4	6	7
القسم الثاني من الصبغيات المنتخبة	7	6	6	7

8. خوارزمية حكم النخبة (Elitism):

في خوارزمية الحالة الثابتة يتم التخلص من الصبغيات ذات درجة الملاءمة السيئة، وانتخاب الصبغيات ذات درجة الملاءمة الجيدة فقط. رغم الميزة الظاهرية المقدمة في خوارزمية الحالة الثابتة، إلا أن فقدان عدد كبير نسبياً من الصبغيات السيئة قد يؤدي إلى الابتعاد أكثر عن الحل الأمثل، وهذا ما يحدث في بعض المسائل كمسألة إيجاد النهاية المحلية العظمى لتابع معطى. حيث تكون

درجة ملاءمة أحد الصبغيات سيئة، لكن إحداثياته قريبة جداً من الحل الأمثل لوجود انحدار شديد بينه وبين الحل الأمثل، فتغير قيمة مورثة واحدة قد ترفع درجة ملاءمة الصبغي بشكل كبير.

تتصف خوارزمية حكم النخبة بأخذ الميزة الإيجابية في خوارزمية الحالة الثابتة، مع حذف سلبيتها. حيث تنفذ بانتخاب الصبغيات الجيدة أولاً، ومن ثم انتخاب بقية الصبغيات من مجموعة صبغيات الجيل كاملة، وليس من مجموعة الصبغيات الجيدة فقط، كما هو الحال في خوارزمية الحالة الثابتة، وبالتالي الإبقاء على فرصة انتخاب صبغيات سيئة. إن عدد الصبغيات المنتخبة في القسم الأول يحدده المستخدم في مرحلة التأهيل الابتدائي، إما كقيمة مطلقة أصغر من حجم الجيل، أو كنسبة مئوية من حجم الجيل. كما أن طريقة انتخاب القسم الثاني هي من خيارات المستخدم، التي يحددها في نظامه البرمجي عند اختياره خوارزمية انتخاب حكم النخبة، فمن الممكن اختيار خوازمية العجلة المتدحرجة أو غيرها من خوارزميات الانتخاب. إن حكم النخبة يسرع عملية الإنتاج بواسطة الخوارزميات الجينية لأنها تمنع فقدان الحلول الجيدة، ولنوضح ذلك بمثال عددي.

مثال:

نفترض جيلاً مكوناً من ثمانية صبغيات لها قيم الملاءمة المبينة في الجدول الآتي:

الصبغي	1	2	3	4	5	6	7	8
درجة الملاءمة	11	10	8	11	10	20	15	5

نفترض كما الأمثلة السابقة، أن الصبغي ذي درجة الملاءمة الأكبر هو الأفضل.

ينتخب القسم الأول بانتخاب عدد من الصبغيات الجيدة، ولتكن النسبة المئوية المدخلة من قبل المستخدم في مرحلة التأهيل الابتدائي هي (25%) من حجم الجيل أي صبغيين فقط. لذلك ينتخب أفضل صبغيين من الجيل الحالي، وهذا ما يتضمنه الجدول الآتي:

القسم الأول من الصبغيات المنتخبة	6	7
درجة الملاءمة	20	15

لانتخاب القسم الثاني فإننا ننفذ خوارزمية الانتخاب المحددة من قبل المستخدم في مرحلة التأهيل الابتدائي كخوارزمية انتخاب للقسم الثاني في خوارزمية حكم النخبة. لتكن الخوارزمية المدخلة هي الانتخاب الإجباري، ولننتخب عدداً من الصبغيات يكمل النقص في حجم الجيل، والذي يحسب بطرح عدد صبغيات القسم الأول من حجم الجيل، وبالتالي ستة صبغيات.

نحسب احتمال انتخاب صبغيات كامل الجيل لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6	7	8
احتمال الانتخاب	0.12	0.11	0.09	0.12	0.11	0.22	0.17	0.06

نحسب القيمة المتوقعة لطلب الصبغي (m)، كما نأخذ القسم الصحيح والقسم الكسري لنحصل على الجدول الآتي:

الصبغي	1	2	3	4	5	6	7	8
القيمة (m)	0.96	0.88	0.72	0.96	0.88	1.76	1.36	0.48
القسم الصحيح	0	0	0	0	0	1	1	0
القسم الكسري	0.96	0.88	0.72	0.96	0.88	0.76	0.36	0.48

الآن، لانتخاب صبغيات القسم الثاني الستة، فإنه ينتخب الصبغي ذي القسم الصحيح الأكبر من الواحد أو يساويه، وبالتالي ينتخب الصبغي السادس والسابع. يستكمل النقص بـترتيب الصبغيات من جديد بشكل تنازلي على حسب القسم الحقيقي للقيمة المتوقعة لطلب الصبغي (m)، ومن ثم يؤخذ أول صبغي ثم الذي يليه حتى الوصول للعدد المطلوب للإكمال والذي أصبح أربعة صبغيات، وذلك وفق الجدول الآتي:

الصبغي المرتب	1	4	2	5	6	3	8	7
القسم الكسري	0.96	0.96	0.88	0.88	0.76	0.72	0.48	0.36

عند تساوي القسم الكسري فيمكن أخذ الصبغيات وفق ترتيب دليلها في المصفوفة المتضمنة لها، فمن المعتاد استخدام المصفوفات في التعامل مع الصبغيات والأجيال. نختار الصبغي الأول والرابع والثاني والخامس حسب تناقص القسم الكسري للقيمة المتوقعة لطلب الصبغي (m)، لتصبح الصبغيات المنتخبة كاملة هي الصبغي السادس والسابع كقسم أول، والسادس والسابع كمرحلة أولى من القسم الثاني، والأول والرابع والثاني والخامس كمرحلة ثانية من القسم الثاني، كما هو موضح في الجدول الآتي:

الصبغي المنتخب	1	2	4	5	6	6	7	7

6-13- العبور (Crossover):

من العمليات الهامة التي تحاكي عملية التزاوج البيولوجي بين الأحياء . فالمعتقد السائد هو أن التزاوج بين أفراد يتمتعون بمواصفات جيدة سوف ينتج عنه أفراداً يتمتعون بمواصفات جيدة على أقل تقدير. فعملية الاندماج بين فردين تم اختيارهما بمستوى ملاءمة جيد يتوقع أن ينتج عنهما سلالة أقوى، أي أن الهدف من هذه العملية هي تأكيد المزج والخلط بين أفراد الجيل الحالي لتحسين النسل الناتج.

تنفذ إجرائية العبور في الخوارزميات الجينية على مرحلتين، الأولى تسمى عملية المزج (Shuffling)، وهي اختيار صبغيات (أفراد) معينة من الجيل المنتخب بإجرائيات الانتخاب السابقة على حسب محدد جديد هو احتمالية العبور ((Probability of crossover (Pc)، يحدد مسبقاً من قبل المستخدم في مرحلة التأهيل الابتدائي. تسمى هذه الصبغيات الجديدة المنتخبة حسب احتمالية العبور

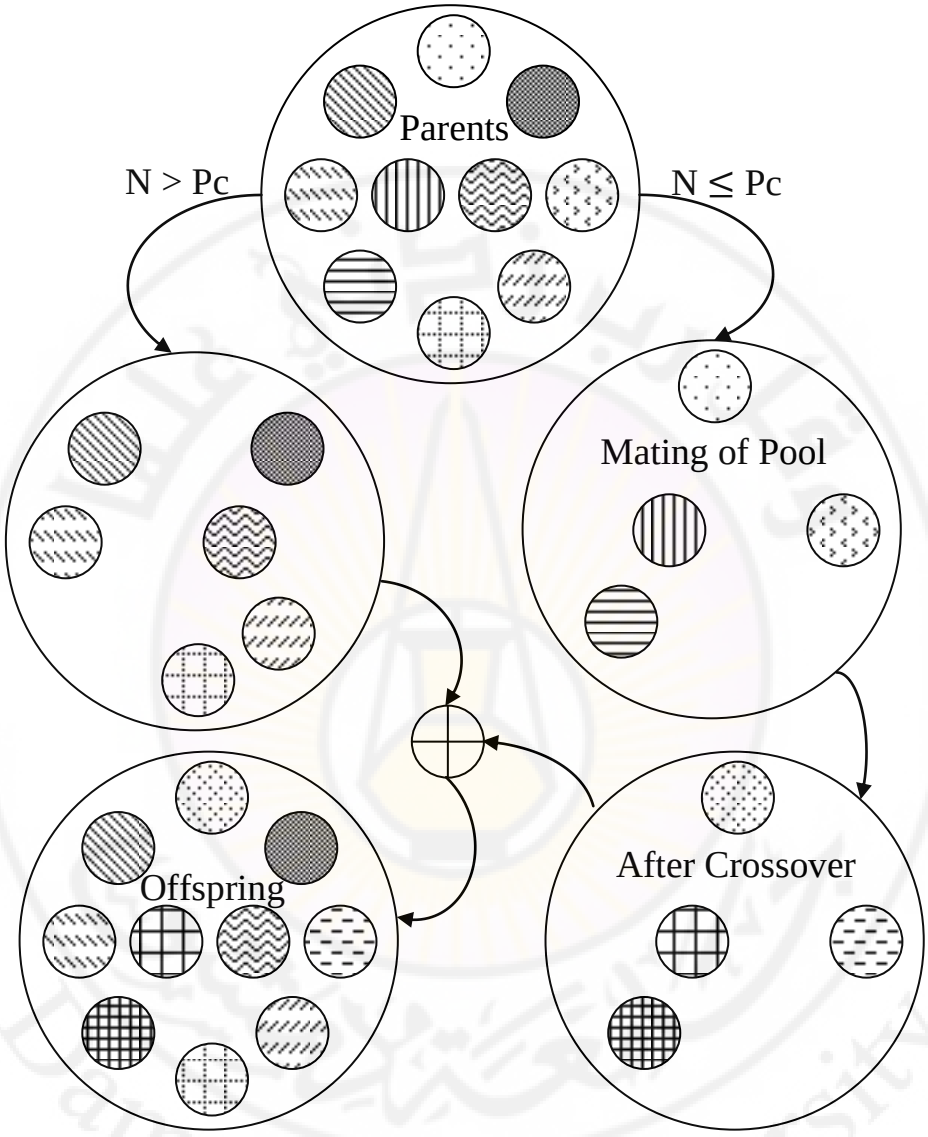
بأعضاء بركة الزمالة (Mating of pool)، أو بركة الجينات (gene of pool). يخلط أعضاء البركة مع بعضهم البعض للحصول على مزج جيد، وبعد عملية المزج نصل إلى المرحلة الثانية ، وهي إجراء عملية العبور بين أعضاء البركة بواحدة من طرائق (خوارزميات) العبور التي سنتناولها بالشرح.

إذن، يكون الجيل بعد العبور عبارة عن قسمين:

القسم الأول: ينتج بعد تنفيذ العبور على بركة الزمالة.

القسم الثاني: ينتقل مباشرة من الجيل المنتخب بعد خوارزميات الانتخاب.

يبين الشكل(6-18) مخططاً توضيحياً لعملية العبور.



الشكل(6-18) تمثيل عملية العبور

لنبين بالتفصيل كلا مرحلتي العبور:

1. الخلط (Shuffling):

يتم أولاً تكوين بركة الصداقة أو الزمالة حسب احتمالية العبور، بتوليد عدد عشوائي حقيقي موجب أصغر أو يساوي الواحد لكل صبغي، فإذا كان العدد العشوائي المولد أصغر أو يساوي قيمة احتمالية العبور، فإنه يتم اختيار الصبغي ضمن البركة. أما إذا كان العدد العشوائي المولد أكبر تماماً من قيمة احتمالية العبور، فإنه لا يتم اختياره. يجب في النهاية أن يكون عدد الصبغيات المختارة زوجياً، وإذا لم يكن ذلك ، نختار صبغياً عشوائياً من الصبغيات غير المختارة، ويضم إلى بركة الزمالة.

تهدف عملية الخلط لتأكيد المزج والخلط، حيث أنه يكثر انتخاب الصبغي نفسه أكثر من مرة في طرائق الانتخاب السابقة ، مثل التسلسل الباقي بدون استبدال، والتسلسل الباقي مع استبدال. فإذا أجريت عملية العبور على الصبغيات المنتخبة، فقد يكثر إجراء العبور بين الصبغي ونفسه، مما لا يحقق الفائدة من عملية العبور، لذلك تجرى عملية الخلط لتحقيق بعض العشوائية.

إحدى طرائق الخلط المستخدمة هي طريقة الخلط الموجودة في أوراق اللعب، حيث يقسم أعضاء البركة إلى مجموعتين، ثم تجري عملية الخلط بوضع واحدة من المجموعة الأولى فوق واحدة من المجموعة الثانية، وهكذا دواليك.

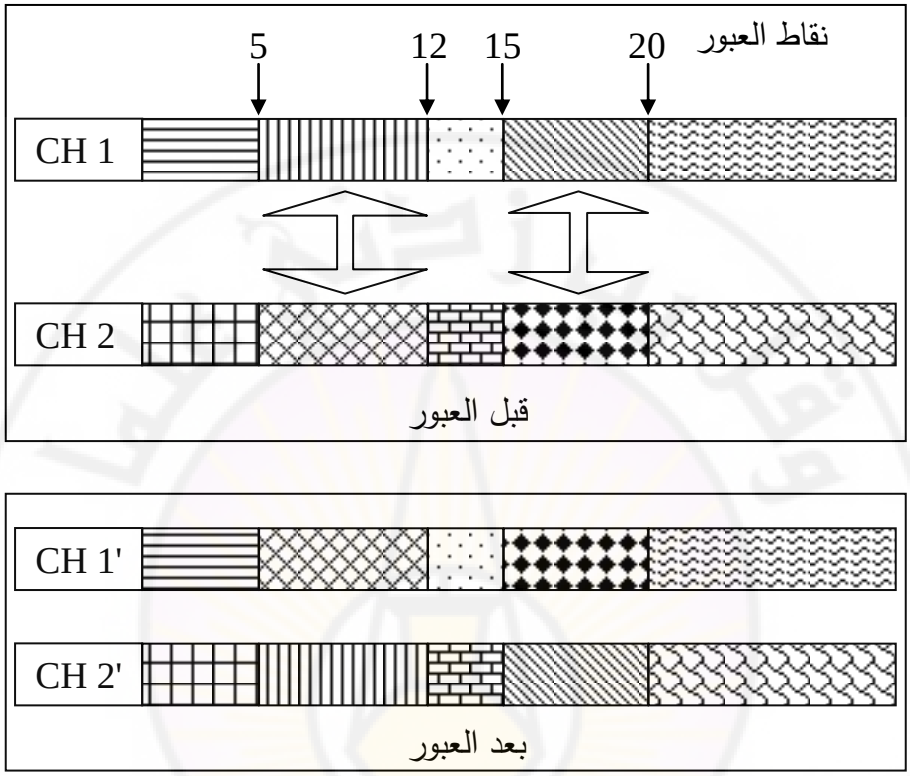
2. طرائق العبور (Crossover Methods):

يوجد الكثير من خوارزميات العبور التي تشترك جميعها بنقل مورثات جيل السلف إلى جيل الخلف. سوف نستعرض أ هم طرائق العبور المستخدمة في الخوارزميات الجينية؛ وهي:

1) العبور البسيط (n) نقطة (Simple n-point crossover):

يتم اختيار (n) نقطة عبور بشكل عشوائي على طول الصبغي، ويجري التبديل بين جميع الصبغيات الوراثية بين نقطة العبور الفردية (i) ضمناً ونقطة العبور الزوجية التالية (i+1) ضمناً أيضاً ، وفي حال كان عدد نقاط العبور (n) فردياً، فإنه يتم تبديل الصبغيات الوراثية بين نقطة العبور الأخيرة (n) حتى نهاية طول الصبغي. ليس من الضروري اختيار نقاط العبور في جميع الصبغيات نفسها، بل تختار نقاط العبور لكل صبغيين متعاقبين ضمن الجيل بشكل متساوٍ وعشوائي. يراعى أن لا تزيد قيمة (n) عن حجم الجيل، وهي من المحددات المدخلة من قبل المستخدم في مرحلة التأهيل الابتدائي عند اختيار خوارزمية العبور البسيط (n) نقطة.

يمكن توضيح هذا النوع من العبور بالشكل(6-19)، وذلك باعتبار أن العبور من النوع البسيط (4-point). يتم اختيار أربع نقاط عشوائية مختلفة على طول الصبغي. تبادل بعد ذلك، المورثات الموجودة بين النقطة الأولى والثانية فيما بين الصبغيين، وبين النقطة الثالثة والرابعة أيضاً. إن القيم العشوائية للنقاط الأربع في مثالنا هي (5) و(15) و(12) و(20). سميت صبغيات جيل الخلف بإضافة إشارة تنصيص للإشارة إلى نشوء صبغيات جديدة مختلفة عن جيل السلف.



الشكل(6–19) عبور (4–point) البسيط

كما يبين الشكل(6–20)، عبوراً بسيطاً بنقطتين أي من النوع (2–point) بقيم (2,6). بهدف التوضيح، فقد وضعت مورثات الصبغي الأول في جيل السلف بخط مائل، ومورثات الصبغي الثاني في جيل السلف بشكل عادي تحته خط، أما المورثات في جيل الخلف فكتبت بشكل عادي.

جيل الآباء	***Chromosome 1***	***1***	***1***	***0***	***1***	***1***	***1***	***1***	***0***
	Chromosome 2	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**
		↓	↓↑	↓↑	↓↑	↓↑	↓↑	↓	↓
جيل الأبناء	**Chromosome 1'**	***1***	**0**	**0**	**0**	**1**	**0**	***1***	***0***
	Chromosome 2'	**0**	***1***	***0***	***1***	***1***	***1***	**0**	**1**

الشكل(6-20) عبور (2-point) البسيط في الترميز الثنائي

2) العبور المنسق (Uniform Crossover):

إن العبور في الطبيعة يكون عشوائياً أكثر من مبدأ العبور (n-point) البسيط، لذا فقد اقترح العبور المنسق الذي تسند فيه كل المورثات من الأب إلى الابن حسب محدد جديد هو احتمالية العبور المنسق (Pu). أبسط حالة في هذا العبور هي محاكاة لعبة (صورة – نقش) التي تلعب بالعملات المعدنية . تحاكى اللعبة بتوليد رقم عشوائي صحيح ضمن المجال [0,1]، أي توليد إما الرقم صفر أو واحد باحتمال متساوٍ، فإذا كان الرقم المولد يساوي الواحد فرضاً، فإن المورثة (الصبغة الوراثية) لدى ا لأب الأول تنتقل إلى الابن الأول ، والمورثة لدى ا لأب الثاني تنتقل إلى الابن الثاني. أما إذا كان الرقم المولد يساوي الصفر، فإن المورثة لدى الأب الأول تنتقل إلى الابن الثاني، والمورثة لدى الأب الثاني تنتقل إلى الابن الأول.

بالطبع، لا يشترط أن تكون قيمة العدد المولد إما صفراً أو واحداً فقط، بل يمكن أن تأخذ أية قيمة احتمالية، حيث تنفذ عملية العبور فقط إذا كان العدد المولد عشوائياً أصغر أو يساوي احتمال العبور المنسق.

يبين الشكل(6-21)، عبوراً منسقاً عند كون احتمال العبور ثنائياً. قمنا بتسمية صبغيات جيل السلف بالآباء (الوالدين) وجيل الخلف بالأبناء بهدف تنويع الأمثلة لتغطي عدة مصطلحات مستخدمة في المراجع.

الرقم العشوائي المولد	1	0	1	0	1	0
Parent 1	*0*	*1*	*1*	*0*	*1*	*0*
Parent 2	1	0	1	0	1	1
Child 1	*0*	0	*1*	0	*1*	1
Child 2	1	*1*	1	*0*	1	*0*

الشكل(6-21) العبور المنسق ذي الاحتمال الثنائي

3) عبور أسوأ مورثة (Cut on worst gene Crossover):

تعتمد خوارزمية العبور (COWGC) على تبادل المورثات عند أسوأ مورثة بين الوالدين (the worst gene). إن المورثة الأسوأ هي المورثة التي تعطي أعلى كلفة، والتي يختلف تعريفها حسب المسألة المدروسة، ولكنها تكون سبباً في زيادة كلفة تابع الملاءمة. فإذا كان تابع الملاءمة يتناهى للصفر عند الاقتراب من الحل الأمثل، فالمورثة الأسوأ تسبب زيادة في قيمة تابع الملاءمة وبالتالي ابتعاد الحل عن الحل الأمثل، والعكس بالعكس. ففي مسألة البائع المتجول تكون المورثة الأسوأ هي المدينة الأكثر بعداً عن المدينة الموجودة على يسارها من بين جميع المدن ضمن الصبغي. أما في مسألة (Knapsack Problem) فإن المورثة الأسوأ هي المورثة التي تشير إلى الغرض ذي أدنى

نسبة بين القيمة والوزن. بما أنه ليس من الضروري أن يتطابق دليل المورثة الأسوأ في كلا الأبوين فإنه يتم إيجاد المورثة الأسوأ في كل والد، ومن ثم اختيار الأسوأ بينهما، ليتم اختيار نقطة القطع (Cut Point (CP)) عند دليل هذه النقطة، وإجراء تبادل للمورثات بين كلا الأبوين بعد هذه النقطة، مع مراعاة ظروف وقيود كل مسألة.

تحسب نقطة القطع (CP) للصبغي (C) ذي الطول (n) في حال المسائل التي يتناهى فيها تابع الملاءمة إلى الصفر (Minimization Problem) بالشكل:

$$CP = arg\max_{1 \le i < n} (Distance(C[i], C[i+1])$$

أما في حال المسائل التي يتناهى فيها تابع الملاءمة إلى قيمة عليا (Maximization Problem) فتحسب بالشكل:

$$CP = arg\min_{1 \le i < n} (Distance(C[i], C[i+1])$$

يبين الشكل(6-22) عملية العبور بين صبغيين، حيث أوجد دليل المورثة الأسوأ في كل صبغي، ومن ثم اختير الدليل الأسوأ ليكون هو نقطة القطع التي تتبادل بعدها المورثات بين الصبغيين.

جيل الآباء	Worst Gene					⤋			
	Chromosome 1	***1***	***1***	***0***	***1***	***1***	***1***	***1***	***0***
	Worst Gene				⤋				
	Chromosome 2	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**
Cut Point					⤊				
		↓	↓	↓	↓↑	↓↑	↓↑	↓↑	↓↑
جيل الأبناء	**Chromosome 1'**	***1***	***1***	***0***	**0**	**1**	**0**	**0**	**1**
	Chromosome 2'	**0**	**0**	**0**	***1***	***1***	***1***	***1***	***0***

الشكل(6-22) عبور أسوأ مورثة

3. عملية العبور في طرائق الترميز الأخرى:

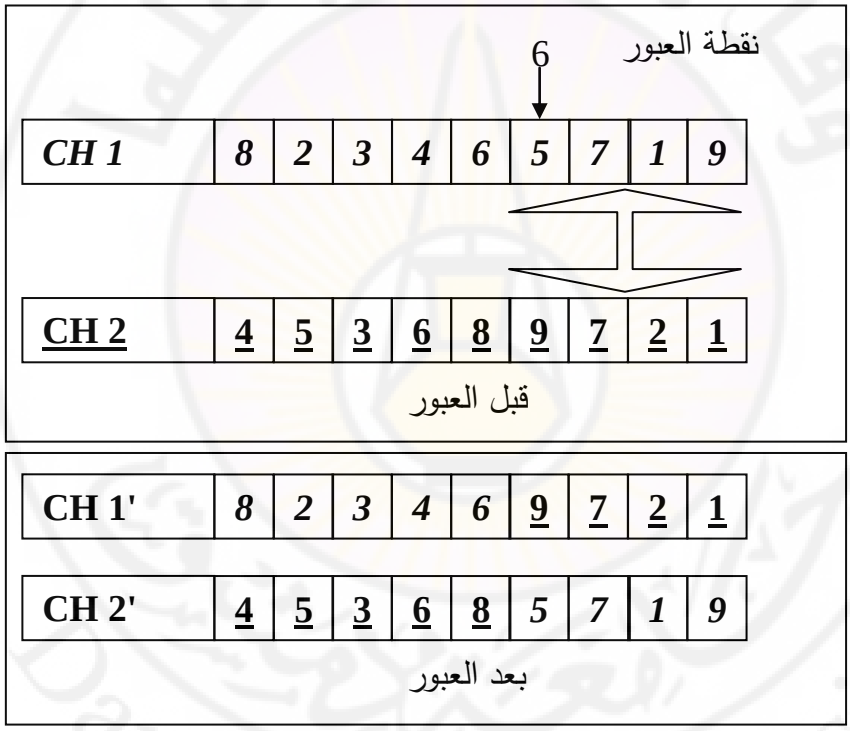
بعد أن تعرفنا على بعض خوارزميات العبور وكيفية تنفيذها عند ترميز الصبغيات ثنائياً، فإننا نستعرض في هذه الفقرة عملية العبور في بقية أنواع الترميز.

1) العبور في الترميز التبديلي (Permutation Encoding Crossover):

لا تتكرر المورثات في الترميز التبديلي (العشري) ضمن الصبغي الواحد، ما ينتج عنه صعوبة في تنفيذ عملية العبور، حيث يجب مراعاة عدم تكرار المورثة وبالتالي وجوب ورود جميع المورثات ضمن الصبغي. يظهر الشكل(6-23)،

حالة تنفيذ العبور أحادي النقطة بشكل مباشر على صبغيين مرمزين عشرياً، حيث تبادل المورثات ابتداءً من نقطة العبور حتى آخر الصبغي.

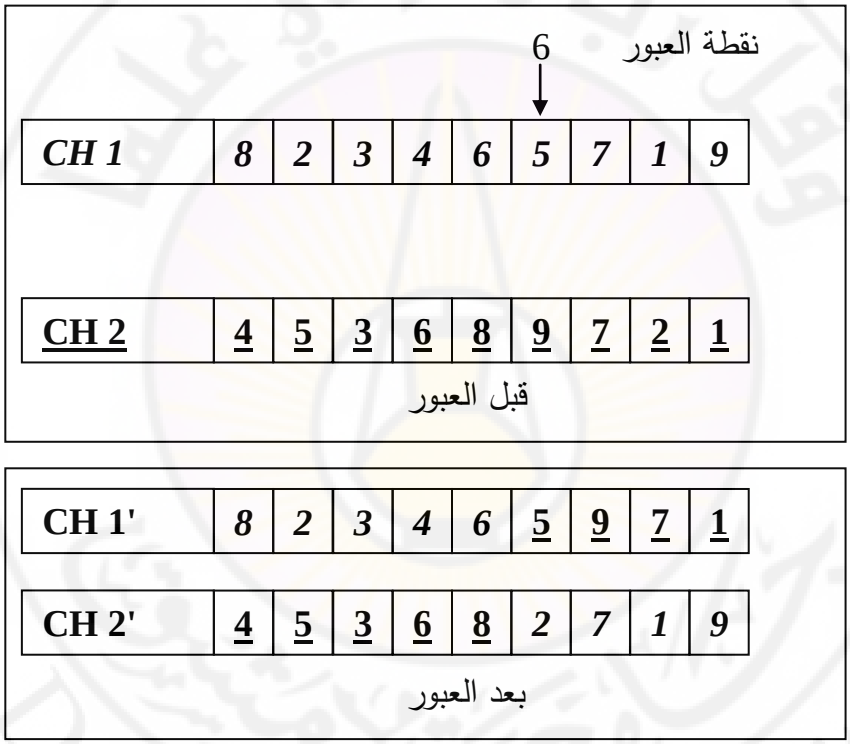
من الملاحظ الخطأ في تنفيذ عملية العبور فقد تكررت بعض المورثات في صبغيات جيل السلف، بالإضافة إلى عدم ورود جميع المورثات، وهي الأرقام الصحيحة ضمن المجال [1, 9] في هذا المثال.



الشكل(6-23) العبور الخاطئ في الترميز العشري

لتنفيذ عملية العبور البسيط في الترميز العشري فإنه يجب إجراء بعض التعديل الذي يحقق عدم التكرار في المورثات. بشكل مشابه لحالة الترميز الثنائي ، ينسخ الصبغي من بداية الأب الأول حتى نقطة العبور، ثم يتابع وضع بقية

المورثات بمسح (Scan) الأب الثاني من بدايته لتضاف كل مورثة مسحت وغير موجودة بعد في الصبغي الابن . يظهر الشكل(6-24)، حالة تنفيذ العبور أحادي النقطة بشكل صحيح على صبغيين مرمزين عشرياً.



الشكل(6-24) العبور الصحيح في الترميز العشري

ليست هذه الطريقة الوحيدة لتنفيذ عملية العبور في الترميز العشري، فهناك عدة خوارزميات أخرى سوف نستعرض بعضاً منها؛ فيما يأتي:

1) خوارزمية التقاطع الجزئي (Partially Matched Crossover ((PMX)):

من الخوارزميات التي تفوقت في أدائها على كثير من خوارزميات العبور في عدة مسائل متباينة. تعتمد على نسخ مقطع من الصبغي الأب الأول إلى الابن الأول مباشرة، ومن ثم ينثر المقطع المقابل في الصبغي الأب الثاني إلى الابن الأول، ويستكمل ما تبقى من مورثات من الأب الثاني.

يمكن توضيح طريقة العمل بالخوارزمية اللغوية الآتية:

1. Randomly select a swath of alleles from parent 1 and copy them directly to the child. Note the indexes of the segment.
2. Looking in the same segment positions in parent 2, select each value that **hasn't already been copied** to the child.
   - ***A****. For each of these values:*
     - ***i.*** *Note the index of this value in Parent 2. Locate the value, V, from parent 1 in this same position.*
     - ***ii.*** *Locate this same value in parent 2.*

iii. *If the index of this value in Parent 2 is part of the original swath, go to step i. using this value.*

iv. *If the position isn't part of the original swath, insert Step A's value into the child in this position.*

Copy any remaining positions from parent 2 to the child.

لتوضيح طريقة العمل، يمكن فرض الصبغيين المرمزين عشرياً المبينين، ولنوجد الأبناء بعد عملية العبور بخوارزمية (PMX).

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

يتم البدء بإيجاد كل صبغي ابن على حدة، ولنبدأ بالابن الأول.

في الخطوة الأولى، يولد عددان عشوائيان مختلفان، قيمهما بين الواحد وطول الصبغي، كدليلين لتحديد بداية ونهاية المقطع، وليكن العددين (4,8). تنسخ مورثات المقطع من الأب الأول إلى الابن الأول.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 1 = – – – *8 0 9 1 3* – –

الخطوة الثانية هي لنثر المورثات الموجودة في مقطع الأب الثاني وغير موجودة في الابن. نبحث عن جميع المورثات الموجودة في المقطع

المقابل في الأب الثاني وغير موجودة في الابن الأول. من الواضح أن المورثات المطلوبة هي (5,2). نبدأ بحلقة تكرارية لمسح (scan) جميع هذه المورثات.

ننطلق من المورثة (5) في الأب الثاني، وننظر للمورثة المقابلة لها في الأب الأول وهي (9). نبحث عن مكان ورود المورثة (9) في الأب الثاني، فنجدها في الموضع الثالث أي ليست ضمن المقطع السابق، لذلك تنقل المورثة (5) إلى هذا الموضع.

Child 1 = – – 5 *8 0 9 1 3* – –

نتابع الحلقة بالانتقال إلى المورثة (2) في الأب الثاني، وننظر للمورثة المقابلة لها في الأب الأول وهي (3). نبحث عن مكان ورود المورثة (3) في الأب الثاني، فنجدها ضمن المقطع السابق، لذلك نعود وننظر للمورثة المقابلة لها في الأب الأول وهي (1). نبحث عن مكان ورود المورثة (1) في الأب الثاني، فنجدها ضمن المقطع السابق أيضاً، لذلك نعود وننظر للمورثة المقابلة لها في الأب الأول وهي (8). نبحث عن مكان ورود المورثة (8) في الأب الثاني، فنجدها في الموضع العاشر أي ليست ضمن المقطع السابق، لذلك تنقل المورثة (2) إلى هذا الموضع.

Child 1 = – – 5 *8 0 9 1 3* – 2

انتهت الحلقة التكرارية، لذلك ننتقل للخطوة الثالثة؛ وهي نسخ ما تبقى من مورثات في الأب الثاني إلى الابن.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 1 = 6 4 5 *8 0 9 1 3* 7 2

لإيجاد الابن الثاني، ما علينا سوى إعادة تكرار الخوارزمية بمراعاة أخذ الأب الثاني دور الأب الأول ولنوضح ذلك بشكل مختصر.

لدينا الصبغيان الآتيان:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

لنفرض عددين عشوائيين جديدين (3,6). تنسخ مورثات المقطع من الأب الثاني لنحصل على الابن المبين:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = - - 9 1 0 5 - - - -

المورثات الموجودة في المقطع المقابل في الأب الأول وغير موجودة في الابن الثاني هي (4,8). نبدأ بحلقة تكرارية لمسح (scan) جميع هذه المورثات.

ننطلق من المورثة (4) حيث تنقل إلى الموضع المبين:

Child 2 = - *4* 9 1 0 5 - - - -

نتابع الحلقة بالانتقال إلى المورثة (8) حيث تنقل إلى الموضع المبين:

Child 2 = - *4* 9 1 0 5 *8* - - -

انتهت الحلقة التكرارية، لذلك ننتقل للخطوة الثالثة؛ وهي نسخ ما تبقى من مورثات في الأب الأول إلى الابن:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = *2 4 9 1 0 5 8 3 7 6*

عند كتابة إجرائية برمجية لهذه الخوارزمية فمن الممكن أن يكون دخلها الوالدان وخرجها الابن. فعند استدعاء الإجرائية لأول مرة، نحصل على الابن الأول. بإعادة استدعاء الإجرائية ثانية، لكن بمبادلة أماكن الوالدين، نحصل على الابن الثاني.

2) خوارزمية العبور الدوري ((Cycle Crossover (CX)):

تعتمد خوارزمية (CX) على القيام بعدد من الدورات بين الصبغيين الوالدين. تنسخ مورثات الدورة الأولى من الأب الأول للابن الأول، ومن الأب الثاني للابن الثاني. بعد ذلك، تنسخ مورثات الدورة الثانية من الأب الأول للابن الثاني، ومن الأب الثاني للابن الأول. يعاد بعد ذلك في الدورة الثالثة نسخ المورثات كما في الدورة الأولى. وهكذا تتعاقب الدورات وعمليات النسخ حتى الوصول لآخر دورة.

لتوضيح طريقة العمل، يمكن فرض الصبغيين المرمزين عشرياً المبينين، ولنوجد الأبناء بعد عملية العبور بخوارزمية (CX).

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

في الدورة الأولى، نبدأ من المورثة الأولى في الأب الأول أي (2). ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (6). نبحث عن (6) في الأب الأول فنجدها في الموضع الأخير. ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (8). نبحث عن (8) في الأب الأول فنجدها في الموضع الرابع. ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (1). نبحث عن (1) في الأب الأول فنجدها في الموضع السابع. ننتقل للمورثة

ذات الدليل نفسه في الأب الثاني وهنا (3). نبحث عن (3) في الأب الأول فنجدها في الموضع الثامن. ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (2). وصلنا إلى المورثة التي بدأنا بها (2)، لذلك تنتهي هنا الدورة الأولى وتنقل هذه المورثات إلى الأبناء كما هو مبين:

Parent 1 = – ***5 4*** – ***0 9*** – – ***7*** –

Parent 2 = – **4 9** – **0 5** – – **7** –

Child 1 = ***2*** – – ***8*** – – ***1 3*** – ***6***

Child 2 = **6** – – **1** – – **3 2** – **8**

في الدورة الثانية، نبدأ من المورثة الأولى الجديدة في الأب الأول أي (5). ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (4). نبحث عن (4) في الأب الأول فنجدها في الموضع الثالث. ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (9). نبحث عن (9) في الأب الأول فنجدها في الموضع السادس. ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (5). وصلنا إلى المورثة التي بدأنا بها (5)، لذلك تنتهي هنا الدورة الثانية وتنقل هذه المورثات إلى الأبناء بشكل متعاكس، كما هو مبين:

Parent 1 = – – – – ***0*** – – – ***7*** –

Parent 2 = – – – – **0** – – – **7** –

Child 1 = ***2*** **4** **9** ***8*** – **5** ***1 3*** – ***6***

Child 2 = **6** ***5 4*** **1** – ***9*** **3** **2** – **8**

في الدورة الثالثة، نبدأ من المورثة الأولى الجديدة في الأب الأول أي (0). ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (0). وصلنا إلى

المورثة التي بدأنا بها (0)، لذلك تنتهي هنا الدورة الثالثة وتنقل هذه المورثات إلى الأبناء بشكل مباشر، كما هو مبين:

Parent 1 = – – – – – – – – ***7*** –

Parent 2 = – – – – – – – – **7** –

Child 1 = ***2 4 9 8 0 5 1 3*** – ***6***

Child 2 = **6 *5 4* 1 0 *9* 3 2** – **8**

في الدورة الرابعة، نبدأ من المورثة الأولى الجديدة في الأب الأول أي (7). ننتقل للمورثة ذات الدليل نفسه في الأب الثاني وهنا (7). وصلنا إلى المورثة التي بدأنا بها (7)، لذلك تنتهي هنا الدورة الرابعة وتنقل هذه المورثات إلى الأبناء بشكل متعاكس، كما هو مبين:

Parent 1 = – – – – – – – – – –

Parent 2 = – – – – – – – – – –

Child 1 = ***2 4 9 8 0 5 1 3 7 6***

Child 2 = **6 *5 4* 1 0 *9* 3 2 7 8**

في الدورة الثالثة والرابعة من هذا المثال يوجد مورثة واحدة متطابقة في كلا الوالدين، لذلك التعاكس هو لتعميم الخوارزمية فقط.

بما أنه لم يبق أي مورثة في الآباء، تنتهي الخوارزمية عند الدورة الرابعة.

يمكن توضيح طريقة عمل خوارزمية (CX) بالخوارزمية اللغوية الآتية:

1. Start a cycle from first gene of first parent to first gene of second parent as shown.

2. Identify the gene in the first position of the second parent and move to the corresponding gene in the first parent.

3. Vertically move from the current gene of the first parent to the gene in the second parent.

4. Check whether the gene in the second parent is same as the first gene of the first parent.

   If Yes, go to step 6; If Not, go to step 5.

5. Move to the gene in the first parent corresponding to the current gene in the second parent and go to step 3.

6. Repeat similar steps to obtain the second offspring.

7. Copy the genes present in the cycle of the first parent to the corresponding positions of the first offspring.

8. Copy the genes present in the cycle of the second parent to the corresponding positions of the second offspring.

9. Copy the remaining genes of the second parent to their corresponding positions of the first offspring.

10. Copy the remaining genes of the first parent to their corresponding positions of the second offspring.

11. The current sequence of genes in each of the offspring forms the final corresponding offspring.

3) خوارزمية العبور الترتيبي ((Order_1 Crossover (OX1):

من أبسط أنواع خوارزميات العبور للصبغيات المرمزة عشرياً، وهي مشابهة للطريقة التي استعرضت في بداية فقرة الترميز التبديلي، ما عدا أن متابعة تعبئة المورثات تتم مباشرة من النقطة الأخيرة للابن الأول والأب الثاني.

لتوضيح طريقة العمل، يمكن فرض الصبغيين المرمزين عشرياً المبينين، ولنوجد الأبناء بعد عملية العبور بخوارزمية (OX1).

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

في الخطوة الأولى، يولد عددان عشوائيان مختلفان، قيمهما بين الواحد وطول الصبغي، كدليلين لتحديد بداية ونهاية المقطع، وليكن العددان (4,8). تنسخ مورثات المقطع من الأب الأول إلى الابن الأول، مع حذف هذه المورثات من الأب الثاني لتسهيل العملية.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 - - - 5 - 2 7 -

Child 1 = - - - *8 0 9 1 3* - -

في الخطوة الثانية، تتم عملية متابعة نقل المورثات إلى الابن الأول من يمين المقطع، لكن من الأب الثاني وانطلاقاً من يمين المقطع الموافق له، وذلك بعد فحص وجود المورثة في الابن أم لا. في المثال الحالي، فإن المورثة الأولى الموجودة على يمين المقطع في الأب الثاني هي (7)، وغير موجودة في الابن، لذلك تنقل إلى الابن ومباشرة على يمين المقطع في حال وجود مكان شاغر.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 – – – 5 – 2 – –

Child 1 = – – – *8 0 9 1 3* 7 –

بمتابعة العملية خطوة أخرى، نحصل على النتيجة الآتية:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – 4 – – – 5 – 2 – –

Child 1 = – – – *8 0 9 1 3* 7 6

بمتابعة العملية بعدة خطوات أخرى، نحصل على النتيجة الآتية:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – – – – – 5 – 2 – –

Child 1 = 4 – – *8 0 9 1 3* 7 6

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – – – – – – – 2 – –

Child 1 = 4 5 – *8 0 9 1 3* 7 6

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – – – – – – – – – –

Child 1 = 4 5 2 8 0 9 1 3 7 6

لإيجاد الابن الثاني، ما علينا سوى إعادة تكرار الخوارزمية بمراعاة أخذ الأب الثاني دور الأب الأول ولنوضح ذلك بشكل مختصر.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

يولد عددان عشوائيان لتحديد بداية ونهاية المقطع، وليكن العددان (2,9). تنسخ مورثات المقطع من الأب الثاني إلى الابن الثاني، مع حذف هذه المورثات من الأب الأول لتسهيل العملية.

Parent 1 = – – – 8 – – – – – 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = – 4 9 1 0 5 3 2 7 –

بمتابعة العملية بعدة خطوات أخرى، نحصل على النتيجة الآتية:

Parent 1 = – – – 8 – – – – – –

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = – 4 9 1 0 5 3 2 7 *6*

Parent 1 = – – – – – – – – – –

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = *8* 4 9 1 0 5 3 2 7 *6*

إن خوارزمية (OX1) ذات أداء مرتفع بسبب بساطة العمليات الرياضية فيها، وقد بينت بعض الدراسات أنها من أسرع خوارزميات العبور، حيث أن عدد الأجيال المولدة بوساطتها يكون مئات أضعاف عدد الأجيال المولدة بخوارزميات أخرى، وللفترة الزمنية نفسها، وقد يصل لألف ضعف.
يمكن توضيح طريقة عمل خوارزمية (OX1) بالخوارزمية اللغوية الآتية:

1. Select a random swath of consecutive alleles from parent 1.
2. Drop the swath down to Child 1 and mark out these alleles in Parent 2.
3. Starting on the right side of the swath, grab alleles from parent 2 and insert them in Child 1 at the right edge of the swath.
4. If you desire a second child from the two parents, flip Parent 1 and Parent 2 and go back to Step 1.

4) خوارزمية العبور الترتيبي المتعدد (Order Multiple Crossover):
تتشابه هذه الخوارزمية مع خوارزمية (OX1) فيما عدا وجود عدة مقاطع فيها.
لتوضيح طريقة العمل، يمكن فرض الصبغيين المرمزين عشرياً المبينين، ولنوجد الأبناء بعد عملية العبور بخوارزمية العبور الترتيبي المتعدد.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

تولد عدة أعداد عشوائية مختلفة، قيمها بين الواحد وطول الصبغي، تحدد بداية ونهاية كل مقطع بشكل متتابع، ولتكن الأعداد هي (2,4,6,9).

يوجد مقطعين يتم نسخهما من الأب الأول إلى الابن الأول مع حذف المورثات المتطابقة في الأب الثاني:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 – – – 0 – – 2 – –

Child 1 = – 5 4 8 – 9 1 3 7 –

تتابع عملية نقل المورثات إلى الابن الأول من يمين المقطع الأخير، لكن من الأب الثاني وانطلاقاً من يمين المقطع الأخير أيضاً.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – – – – 0 – – 2 – –

Child 1 = – 5 4 8 – 9 1 3 7 6

بمتابعة العملية بعدة خطوات أخرى، نحصل على النتيجة الآتية:

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – – – – – – – 2 – –

Child 1 = 0 5 4 8 – 9 1 3 7 6

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = – – – – – – – – – –

Child 1 = 0 5 4 8 2 9 1 3 7 6

لإيجاد الابن الثاني، ما علينا سوى إعادة تكرار الخوارزمية بمراعاة أخذ الأب الثاني دور الأب الأول، ولنوضح ذلك بشكل مختصر.

Parent 1 = 2 5 4 8 0 9 1 3 7 6

Parent 2 = 6 4 9 1 0 5 3 2 7 8

تولد عدة أعداد عشوائية مختلفة، ولتكن (1,4,7,10). يوجد مقطعان يتم نسخهما من الأب الثاني إلى الابن الثاني مع حذف المورثات المتطابقة في الأب الأول:

Parent 1 = - 5 - - 0 - - - - -

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = 6 4 9 1 - - 3 2 7 8

بمتابعة العملية بعدة خطوات، نحصل على النتيجة الآتية:

Parent 1 = - - - - 0 - - - - -

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = 6 4 9 1 *5* - 3 2 7 8

Parent 1 = - - - - - - - - - -

Parent 2 = 6 4 9 1 0 5 3 2 7 8

Child 2 = 6 4 9 1 *5 0* 3 2 7 8

5) خوارزمية الحواف المجمعة (Edge Recombination Crossover):

في مسألة البائع المتجول فإن أكثر ما يخفض ملاءمة الصبغي هو إحداث حواف بين عقد شبكة المدن. تنطلق فكرة خوارزمية الحواف من تخفيض الحواف الممكن إحداثها بين المورثات عند القيام بعملية العبور. لتوضيح طريقة العمل، يمكن فرض الصبغيين المرمزين عشرياً المبينين، ولنوجد الأبناء بعد عملية العبور بخوارزمية الحواف.

Parent 1 = A B F E D G C

Parent 2 = G F A B C D E

في الخطوة الأولى، تولد قائمة المورثات المتجاورة، حيث تسرد جميع المورثات في الجيل مع جميع المورثات المجاورة لها في كلا الصبغيين، مع الأخذ بعين الاعتبار تجاور أول مورثة وآخر مورثة في الصبغي.

A: B C F

B: A F C

C: A G B D

D: E G C

E: F D G

F: B E G A

G: D C E F

يتم البدء بابن فارغ من المورثات:

Child 1 = Empty Chromosome

في الخطوة الأولى، يتم اختيار المورثة الأولى من أحد الصبغيين الآباء عشوائياً، ولتكن (A) مثلاً، وتوضع في الابن الأول.

Child 1 = A

تحذف بعد ذلك (A) من قائمة التجاور السابقة لنحصل على القائمة الجديدة الآتية:

–: B C F

B: – F C

C: – G B D

D: E G C

E: F D G

F: B E G –

G: D C E F

تضاف المورثة الأقصر في القائمة إلى الابن الأول، وهنا (B) هي الأقصر، حيث تحتوي مورثتين في قائمتها.

Child 1 = A B

تحذف بعد ذلك (B) من قائمة التجاور الأخيرة لنحصل على القائمة الجديدة الآتية:

–: – C F

–: – F C

C: – G – D

D: E G C

E: F D G

F: – E G –

G: D C E F

تضاف المورثة الأقصر في القائمة إلى الابن الأول، وهنا لدينا مورثتان متساويتان (C) و(F) هما الأقصر، حيث تحتوي كل واحدة منهما على مورثتين. تختار إحدى المورثتين عشوائياً، ولتكن (F).

Child 1 = A B F

تحذف بعد ذلك (F) من قائمة التجاور الأخيرة لنحصل على القائمة الجديدة الآتية:

–: – C –

–: – – C

C: – G – D

D: E G C

E: – D G

–: – E G –

G: D C E –

تضاف المورثة الأقصر في القائمة إلى الابن الأول، وهنا لدينا مورثتان متساويتان (C) و(E) هما الأقصر، حيث تحتوي كل واحدة منهما على مورثتين. تختار إحدى المورثتين عشوائياً، ولتكن (E).

Child 1 = A B F E

تحذف بعد ذلك (E) من قائمة التجاور الأخيرة لنحصل على القائمة الجديدة الآتية:

–: – C –

–: – – C

C: – G – D

D: – G C

–: – D G

–: – – G –

G: D C – –

بالطريقة نفسها يصبح الابن بالشكل:

Child 1 = A B F E G

بحذف (G) تصبح القائمة الجديدة:

–: – C –

–: – – C

C: – – – D

D: – – C

–: – D –

–: – – – –

–: D C – –

وبالتالي، يصبح الابن بالشكل:

Child 1 = A B F E G C

بحذف (C) تصبح القائمة الجديدة:

–: – – –

–: – – –

–: – – – **D**

D: – – –

–: – **D** –

–: – – – –

–: **D** – – –

في حال بقيت مورثات ضمن القائمة ولم تضاف إلى الابن، فإنه يتم اختيارها بترتيب عشوائي وتضاف للابن. في هذا المثال بقيت (D) فقط، لذلك تضاف مباشرة.

Child 1 = A B F E G C D

يلاحظ في النتيجة وجود حافة جديدة واحدة فقط بعد إجراء العبور، وهي الحافة من (A) إلى (D)، وهذا ما يميز خوارزمية الحواف عن غيرها من خوارزميات العبور.

يقابل ميزة التقليل من الحواف الجديدة، زمناً طويلاً في عمليات المعالجة، فخوارزمية الحواف من أبطأ خوارزميات العبور، حيث بينت بعض الدراسات أنها أبطأ من خوارزمية العبور البسيط بنقطة واحدة التي استعرضت في البداية بحوالي (200) مرة.

يمكن توضيح طريقة عمل خوارزمية الحواف بالخوارزمية اللغوية الآتية:

1. X = the first node from a random parent.
2. While the CHILD chromo isn't full, Loop:
   A. Append X to CHILD

B. Remove X from Neighbor Lists

If X's neighbor list is empty:

Z = random node not already in CHILD

Else

Determine neighbor of X that has fewest neighbors

If there is a tie, randomly choose 1

Z = chosen node

X = Z

2) العبور في ترميز القيمة (Value Encoding Crossover):

تختلف المسائل التي تحل باستخدام الخوارزميات الجينية بشكل كبير، كما قد تختلف طرائق الحل وتوابع الملاءمة وطرائق الترميز للمسألة الواحدة. يشكل هذا الأمر تحدياً لوضع قواعد ثابتة لحل المسائل باستخدام الخوارزميات الجينية. لذلك يبقى مصمم النظام البرمجي والباحث هما الأقدر على وضع القواعد المناسبة لكل حالة، فربما حلت المسألة نفسها بأكثر من طريقة. فعلى سبيل المثال، تحل مسألة البائع المتجول وفق طريقة الترميز العشري، ما يفترض عدم تكرار المورثة ووجوب ورود جميع المورثات ضمن الصبغي. ولكن في مسألة أخرى تتطلب إيجاد الأمثال الصحيحة لكثير حدود يمكن استخدام الترميز العشري وعدم اشتراط وجوب ورود جميع المورثات ضمن الصبغي وعدم تكرار المورثات.

كما نعلم فإن ترميز القيمة هو الترميز الأكثر عمومية بين طرائق الترميز الأخرى، ما يعني صعوبة وضع قواعد موحدة لجميع الحالات. بشكل عام، فإن جميع طرائق العبور المستخدمة في الترميز الثنائي يمكن أن تستخدم هنا . يبين

الشكل(6−25) كيفية تنفيذ العبور المنسق باحتمال (Pu=0.3) بين صبغيين مرمزين بالقيمة باستخدام أعداد حقيقية. أما الشكل(6−26) فيبين كيفية تنفيذ العبور البسيط (2−point) بين صبغيين مرمزين بالقيمة باستخدام أحرف اللغة الإنكليزية. نلاحظ أنه يمكن هنا تكرار المورثة، كما أنه ليس من الضروري ورود جميع المورثات في كل صبغي.

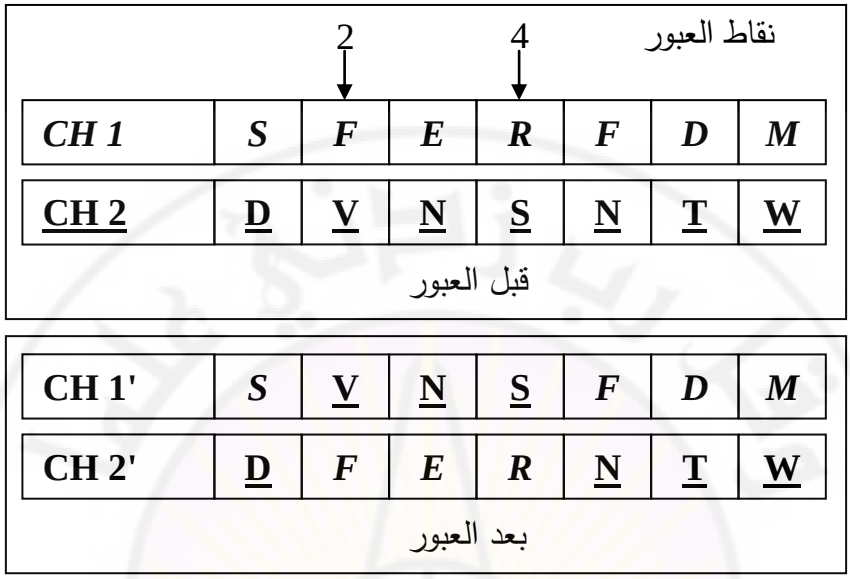
Pu = 0.3	**0.8**	**0.3**	**0.7**	**1.0**	**0.1**	**0.2**	***0.9***
CH 1	***2.4***	***3.1***	***2.8***	***3.9***	***7.1***	***2.4***	***5.8***
CH 2	**5.1**	**6.6**	**8.4**	**2.8**	**4.6**	**5.2**	**1.5**

قبل العبور

CH 1'	***2.4***	**6.6**	***2.8***	***3.9***	**4.6**	**5.2**	***5.8***
CH 2'	**5.1**	**3.1**	**8.4**	**2.8**	***7.1***	***2.4***	**1.5**

بعد العبور

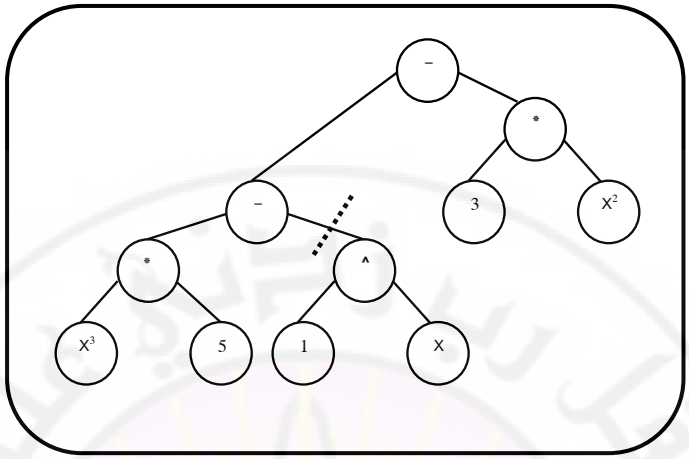
الشكل(6−25) العبور المنسق في ترميز القيمة باستخدام أعداد حقيقية



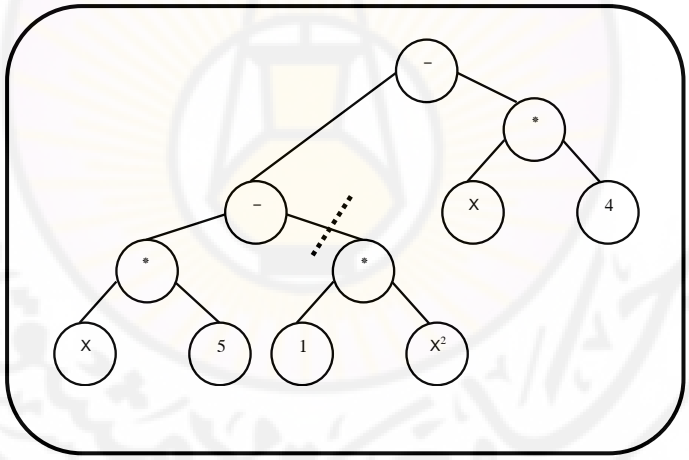
الشكل(6-26) العبور المنسق في ترميز القيمة باستخدام الأحرف الإنكليزية

3) العبور في الترميز الشجري (Tree Encoding Crossover):

إن صعوبة هذا النوع من الترميز، ومحدودية استخدامه، أدى إلى عدم انتشاره بشكل واسع، باستثناء استخدامه في برامج التطوير والخبرة كما في البرمجة الجينية (GP). تنفذ عملية العبور باقتطاع فرع عشوائياً من شجرة الأب الأول ومبادلته بفرع مقابل من شجرة الأب الثاني، ما ينتج عنه الابن الأول والابن الثاني. تبين الأشكال(6-27) و(6-28) و(6-29) و(6-30)، أشجار صبغيات الآباء والأبناء، حيث يلاحظ كيفية اقتطاع الفروع من الآباء ومبادلتها للحصول على الأبناء.

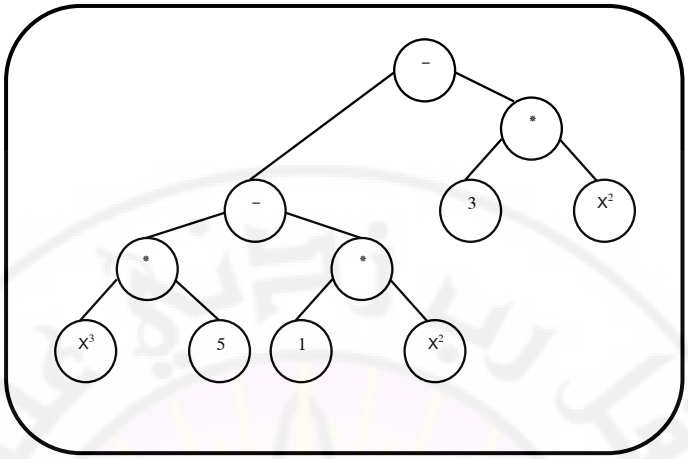


الشكل(6-27) Parent 1

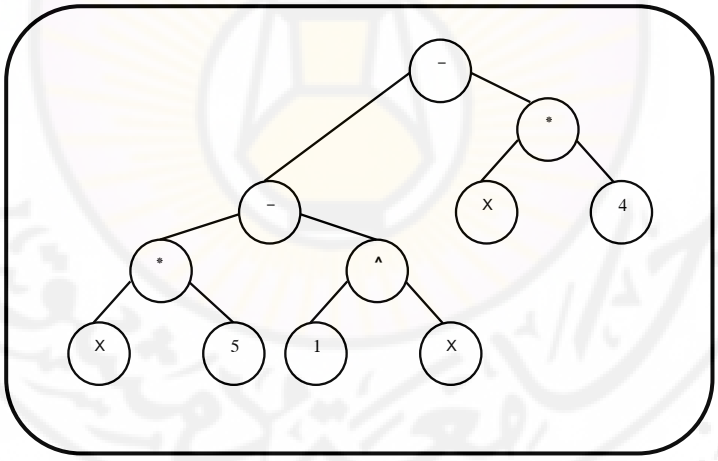


الشكل(6-28) Parent 2

Damascus University



الشكل(6-29) Child 1



الشكل(6-30) Child 2

6-14- الطفرة (Mutation):

تمثل المرحلة الأخيرة من سلسلة العمليات التكرارية والتي تساهم بشكل جيد في الوصول إلى الحل الأمثل بسرعة، وهي مستمدة من كون الخوارزميات الجينية فرعاً من تقنيات البحث العشوائي عن الحل الأمثل. وبالتالي فإن حدوث تغير مفاجئ في الجيل وغير متوقع، أي بشكل عشوائي ولو كان باحتمال حدوث طفيف سوف يكون له الأثر الإيجابي في الاقتراب من الحل الأمثل.

تمنع عملية التطور تصنيف جميع الحلول في أفراد الجيل الحالي كحل أمثل وحيد ضمن نطاق بحث ضيق. يمكن باستخدام فكرة الطفرة الحصول على حلول ملائمة جديدة، ليست موجودة في أفراد الجيل الحالي أو السابق ، إذ قد تكون حلاً ملائماً أكثر من الحل السابق ، وبالتالي التوجه نحو الحل الأمثل ضمن فضاء الحل كاملاً.

إن الطفرة هي عملية عشوائية بنسب احتمالية صغيرة جداً ، إذ يتم تغيير صفات أفراد الجيل الناتج بعد عملية العبور.

تعني الطفرة في حالة الترميز الثنائي أن هناك مورثات من الصبغي تختار بشكل عشوائي، ومن ثم يؤخذ متممها أي تعكس حالتها من "0" إلى "1" وبالعكس.

تعتمد الطفرة بشكل أساسي كما هو الحال في عملية العبور على طريقة ترميز الصبغيات، فمثلاً يمكن أن تكون الطفرة عبارة عن تبديل مواضع مورثتين من الصبغي، أو تبديل مواضع كل المورثات بعد نقطة العبور، أو تبديل كل المورثات بين نقطتي عبور، أو عن طريق تابع عشوائي كلما تحقق شرطه.

إذا لم تحدث الطفرة فإن الابن سوف ينتج عن عملية العبور مباشرة، أو أنه ينتج عن عملية نسخ صبغي الأب (إذا لم تحدث عملية العبور أيضاً) وكذلك بدون أي تغيير، أما إذا حدثت الطفرة فإن جزءاً أو أكثر من الصبغي سوف يتغير.

يعرف محدد جديد في الخوارزميات الجينية هو احتمال الطفرة (Probability of mutation)، ويدل على احتمال حدوث الطفرة [100% - 0%]، فإذا كان احتمال الطفرة (100%)، فإن كل الصبغي سوف يتغير، أما إذا كان الاحتمال (0%)، فلا شيء في الصبغي يتغير.

الهدف من إجراء عملية الطفرة هو توسيع نطاق منطقة البحث ضمن فضاء الحل، كي لا تتركز الحلول ضمن منطقة ضيقة، كحالة وجود نهاية محلية عظمى مع وجود نهاية محلية أكبر قيمة في منطقة أخرى من التابع.

تمنع الطفرة الخوارزميات الجينية من الانزلاق نحو الحل الأفضل الحالي، أي أنها تساهم في جعل أفضل حل في الجيل الجديد أفضل من أفضل حل في الجيل السابق.

يجب أن لا يكون احتمال الطفرة كبيراً، كي لا تتغير الخوارزميات الجينية من بحث عشوائي موجه إلى بحث عشوائي غير موجه عند كل عملية توليد لفرد من الجيل الجديد. وهذا يعني عدم إمكانية التطور باتجاه الحل الأمثل بشكل صحيح.

1. عملية الطفرة حسب نوع الترميز:

كما في العبور فإن نوع الطفرة يعتمد على الترميز وعلى المشكلة بحد ذاتها. يوجد العديد من طرائق الطفرة التي سوف نتناولها باختصار وعبر عدة أمثلة واقتراحات لكيفية تنفيذها مع تعدد أنواع الترميز.

1) الطفرة في الترميز الثنائي (Binary Encoding Mutation):

من أبسط وأسرع عمليات الطفرة وأكثرها انتشاراً. تحدث الطفرة في الترميز الثنائي بإيجاد متمم الرقم الثنائي، أي ببساطة، عكس قيمة الرقم الثنائي المنتخب بشكل منطقي. يبين الشكل(6-31) كيفية حدوث طفرة في صبغي تم ترميزه ثنائياً،

حيث يلاحظ تغيير قيمة الرقم (1) الموجود في الخلية الثالثة من الصبغي إلى قيمة (0).

قبل الطفرة	Chromosome A	1	0	**1**	1	1	1	1	0
		↓	↓	↓↑	↓	↓	↓	↓	↓
بعد الطفرة	Chromosome A'	1	0	**0**	1	1	1	1	0

الشكل(6-31) حدوث طفرة على صبغي ثنائي

بعد التعرف على كيفية حدوث الطفرة في صبغي مرمز ثنائياً ، نبين الخطوات التفصيلية لهذه العملية.

بعد أن تنفذ عملية الانتخاب على الجيل الحالي، تجري عملية العبور بين الصبغيات لنحصل بعدها على جيل الأبناء الابتدائي ما قبل إجراء الطفرة. بعد ذلك، تطبق خوارزمية الطفرة المناسبة للترميز المستخدم في الصبغيات، لنحصل في نهاية العملية على جيل الأبناء النهائي وهو الجيل الجديد الذي سوف يفحص من خلاله شرط التوقف.

ليس من الضروري حدوث طفرة في الجيل، فهذا يتعلق بمحدد احتمال الطفرة (Pm)، وهو من المحددات التي يدخلها المستخدم في مرحلة التأهيل الابتدائي. يدل (Pm) على احتمال حدوث طفرة إما على الجيل أو الصبغي أو المورثة، وهذا يتحدد بنوع الترميز وبما يختاره المستخدم من إعدادات في البيئة

البرمجية. عموماً، عندما يكون احتمال الطفرة كبيراً نسبياً، والتنفيذ على مستوى الجيل، فهذا يكافئ أن يكون احتمال الطفرة صغيراً، والتنفيذ على مستوى المورثة.

لنبين خطوات خوارزمية الطفرة على المستويات الثلاثة:

- على مستوى الجيل:
  - توليد عدد عشوائي حقيقي ضمن المجال [0,1].
  - إذا كان العدد أكبر من احتمال الطفرة (Pm) انتقل للخطوة الأخيرة.
  - إذا كان العدد أصغر أو يساوي احتمال الطفرة (Pm)، فقم بتوليد عدد عشوائي بين الواحد وحجم الجيل، أي ضمن المجال [1,Population Size].
  - قم باختيار الصبغي ذي الترتيب الموافق للعدد العشوائي المولد في الخطوة السابقة.
  - قم بتوليد عدد عشوائي صحيح بين الواحد وطول الصبغي، أي ضمن المجال [1,Chromosome Length].
  - قم بتغيير الرقم الثنائي الموجود في الخانة ذات الدليل الموافق للعدد العشوائي المولد في الخطوة السابقة إلى متممه الثنائي.
  - انتهاء الخوارزمية.

مثال:

لدينا جيل من الصبغيات المرمزة ثنائياً بحجم (Ps=10)، نفذت على الجيل عملية الانتخاب، ومن ثم العبور فحصلنا على الجيل المبين. المطلوب،

أوجد جيل الأبناء بعد تنفيذ مرحلة الطفرة على مستوى الجيل، علماً أن (Pm=0.20).

$$\mathrm{Pop} = \begin{bmatrix} 0&1&1&1&0&1&0&1&1&1&1&1 \\ 0&1&0&0&0&1&1&0&1&0&1&0 \\ 1&1&1&0&0&1&1&1&0&0&0&0 \\ 0&0&1&0&1&1&0&1&1&0&0&1 \\ 0&1&1&0&0&1&0&1&1&0&0&0 \\ 0&1&1&0&1&1&1&0&1&0&0&1 \\ 0&0&1&1&0&1&0&1&1&0&0&1 \\ 0&1&1&1&1&1&0&1&1&0&0&0 \\ 1&1&1&0&0&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&1&1&0&1 \end{bmatrix}$$

يبدأ الحل بتوليد عدد عشوائي ضمن المجال [0,1]، وليكن العدد (0.70). وبما أن العدد أكبر من احتمال الطفرة (Pm) ننتقل للخطوة الأخيرة، وهي الخروج من الخوارزمية دون حدوث طفرة على الجيل.

يمكن توليد عدد عشوائي صحيح ضمن المجال [0,100]، واعتبار احتمال الطفرة مضروباً بمائة، لتسهيل الحسابات الرياضية.

لنفرض أن العدد العشوائي المولد (0.14). بما أن العدد أصغر من احتمال الطفرة (Pm)، نقوم بتوليد عدد عشوائي بين الواحد وحجم الجيل، أي ضمن المجال [1,10].

ليكن العدد العشوائي المولد هو (3). نختار الصبغي ذي الترتيب الموافق للعدد العشوائي المولد، أي الصبغي الثالث في مصفوفة الجيل.

نقوم بعد اختيار الصبغي الواجب تنفيذ الطفرة عليه بتوليد عدد عشوائي بين الواحد وطول الصبغي، أي ضمن المجال [1,12].

ليكن العدد العشوائي المولد هو (4). نختار المورثة ذات الترتيب الموافق للعدد العشوائي المولد، أي المورثة الرابعة في الصبغي الثالث من مصفوفة الجيل ونتمم قيمة هذا الرقم الثنائي، حيث كتبت المورثة المعنية بخط غامق ضمن إطار مغلق، ليكون الجيل الجديد أي جيل الأبناء كما هو مبين.

$$\text{Pop} = \begin{bmatrix} 0\,1\,1\,1\,0\,1\,0\,1\,1\,1\,1\,1 \\ 0\,1\,0\,0\,0\,1\,1\,0\,1\,0\,1\,0 \\ 1\,1\,1\,\mathbf{1}\,1\,0\,1\,1\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,1 \\ 0\,1\,1\,0\,0\,1\,0\,1\,1\,0\,0\,0 \\ 0\,1\,1\,0\,1\,1\,1\,0\,1\,0\,0\,1 \\ 0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,0\,1 \\ 0\,1\,1\,1\,1\,1\,0\,1\,1\,0\,0\,0 \\ 1\,1\,1\,0\,0\,1\,0\,1\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,1 \end{bmatrix}$$

- على مستوى الصبغي:
  - تكرار الخطوات اللاحقة على عدد الصبغيات أي بحجم الجيل، وعند الانتهاء الانتقال للخطوة الأخيرة.
  - توليد عدد عشوائي حقيقي ضمن المجال [0,1].
  - إذا كان العدد أكبر من احتمال الطفرة (Pm) انتقل للخطوة الأولى.

- إذا كان العدد أصغر أو يساوي احتمال الطفرة (Pm)، فقم بتوليد عدد عشوائي صحيح بين الواحد وطول الصبغي، أي ضمن المجال [1,Chromosome Length].
- قم بتغيير الرقم الثنائي الموجود في الخانة ذات الدليل الموافق للعدد العشوائي المولد في الخطوة السابقة إلى متممه الثنائي.
- العودة للخطوة الأولى.
- انتهاء الخوارزمية.

مثال:

لدينا جيل من الصبغيات المرمزة ثنائياً بحجم (Ps=10)، نفذت على الجيل عملية الانتخاب ومن ثم العبور فحصلنا على الجيل المبين. المطلوب، أوجد جيل الأبناء بعد تنفيذ مرحلة الطفرة على مستوى الصبغي، علماً أن (Pm=0.10).

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

نكرر الخطوات اللاحقة على جميع صبغيات الجيل كما يأتي:

يبدأ الحل بتوليد عدد عشوائي ضمن المجال [0,1]، وليكن العدد (0.54). وبما أن العدد أكبر من احتمال الطفرة (Pm) ننتقل للخطوة الأولى، وهي فحص وجود صبغيات أخرى أو انتهاء الخوارزمية.

في حال كان العدد العشوائي (0.09). بما أن العدد أصغر من احتمال الطفرة (Pm)، نقوم بتوليد عدد عشوائي بين الواحد وطول الصبغي، أي ضمن المجال [1,12].

ليكن العدد العشوائي المولد هو (2). نختار المورثة ذات الترتيب الموافق للعدد العشوائي المولد، أي المورثة الثانية في الصبغي الحالي المختبر من مصفوفة الجيل ونتمم قيمة هذا الرقم الثنائي.

بافتراض أن الخوارزمية نفذت وكانت الأعداد العشوائية المولدة بالشكل الآتي:

RN=[0.12 0.34 0.56 0.87 0.07
0.16 0.46 0.02 0.27 0.78]

تحدث الطفرة في الصبغي الخامس والثامن فقط، لأن العدد العشوائي المولد أصغر أو يساوي احتمال الطفرة في الدليل الخامس والثامن ضمن مصفوفة (RN)، وذلك على اعتبار أن الدليل الأول يبدأ من الواحد، كما هو معمول به في بيئة البرمجة (MATLAB)، وليس من الصفر كبعض لغات البرمجة مثل (C).

نبين فيما يأتي الجيل الجديد أي جيل الأبناء بعد إجراء الطفرة على الصبغي الخامس والثامن ، وفي المورثة السابعة والثانية تباعاً، واللتين حددتا عشوائياً أيضاً كما سبق وبينا سابقاً.

$$\text{Pop} = \begin{bmatrix} 0\,1\,1\,1\,0\,1\,0\,1\,1\,1\,1\,1 \\ 0\,1\,0\,0\,0\,1\,1\,0\,1\,0\,1\,0 \\ 1\,1\,1\,0\,0\,1\,1\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,1 \\ 0\,1\,1\,0\,0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,1\,1\,0\,1\,1\,1\,0\,1\,0\,0\,1 \\ 0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,0\,1 \\ 0\,0\,1\,1\,1\,1\,1\,0\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,1\,0\,1\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,1 \end{bmatrix}$$

- على مستوى المورثة:
  - تكرار الخطوات اللاحقة على عدد الصبغيات أي بحجم الجيل، وعند الانتهاء الانتقال للخطوة الأخيرة.
  - من أجل كل مورثة في الصبغي يتم توليد عدد عشوائي حقيقي ضمن المجال [0,1].
  - إذا كان العدد أكبر من احتمال الطفرة (Pm) انتقل للمورثة التالية.
  - إذا كان العدد أصغر أو يساوي احتمال الطفرة (Pm)، فقم بتغيير قيمة المورثة الحالية إلى متممها الثنائي.
  - العودة للخطوة الأولى.
  - انتهاء الخوارزمية.

مثال:

لدينا جيل من الصبغيات المرمزة ثنائياً بحجم (Ps=10)، نفذت على الجيل عملية الانتخاب ومن ثم العبور فحصلنا على الجيل المبين. المطلوب، أوجد جيل الأبناء بعد تنفيذ مرحلة الطفرة على مستوى الصبغي، علماً أن (Pm=0.04).

$$\text{Pop} = \begin{bmatrix} 0&1&1&1&0&1&0&1&1&1&1&1 \\ 0&1&0&0&0&1&1&0&1&0&1&0 \\ 1&1&1&0&0&1&1&1&0&0&0&0 \\ 0&0&1&0&1&1&0&1&1&0&0&1 \\ 0&1&1&0&0&1&0&1&1&0&0&0 \\ 0&1&1&0&1&1&1&0&1&0&0&1 \\ 0&0&1&1&0&1&0&1&1&0&0&1 \\ 0&1&1&1&1&1&0&1&1&0&0&0 \\ 1&1&1&0&0&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&1&1&0&1 \end{bmatrix}$$

يبدأ الحل بالمرور على جميع المورثات في كل صبغي، وتوليد عدد عشوائي ضمن المجال [0,1]، فإذا كان العدد أكبر من احتمال الطفرة (Pm) ننتقل للمورثة التالية، أو للمورثة الأولى من الصبغي التالي عند الوصول لنهاية الصبغي.

في حال كان العدد العشوائي أصغر أو يساوي احتمال الطفرة (Pm)، نقوم بإتمام قيمة المورثة الثنائية.

بافتراض أن الخوارزمية نفذت، وحصلنا على النتيجة الآتية:

$$\mathrm{Pop} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

من الواضح، أنه حدث أربع طفرات في ثلاثة صبغيات.

في الطفرة على مستوى المورثة، يتم توليد عدد من الأعداد العشوائية يساوي إلى حجم الجيل مضروباً بطول الصبغي، وذلك على عدد المورثات الموجودة في الجيل.

2) الطفرة في الترميز التبديلي (Permutation Encoding Mutation):

لا يصح في الترميز التبديلي إجراء الطفرة على مورثة واحدة فقط ضمن الصبغي، لأن ذلك يعني تكراراً لمورثة ما، وهذا غير مسموح في الترميز التبديلي الذي يشترط عدم تكرار المورثات، ووجوب ورود جميع المورثات في كل صبغي، ولكن بترتيب مختلف، كمسألة البائع المتجول.

تحدث الطفرة بعدة طرائق كاختيار مورثتين عشوائياً من مورثات الصبغي، والتبديل بينهما. وهذا يضمن عدم التكرار، وورود جميع المورثات في الصبغي.

يبين الشكل(6–32) كيفية حدوث طفرة في صبغي تم ترميزه عشرياً، حيث يلاحظ تبديل المورثتين في الموضع الثالث والسادس.

قبل الطفرة	Chromosome A	5	6	**2**	1	8	**4**	3	7
		↓	↓	↔	↓	↓	↔	↓	↓
بعد الطفرة	Chromosome A'	5	6	**4**	1	8	**2**	3	7

الشكل(6–32) حدوث طفرة على صبغي مرمز عشرياً

بشكل مشابه للطفرة في الترميز الثنائي فإنه ليس من الضروري حدوث طفرة في الجيل، فهذا يتعلق بمحدد احتمال الطفرة (Pm). يدل (Pm) على احتمال حدوث طفرة إما على الجيل أو الصبغي، وليس على مستوى المورثة، كما في الترميز الثنائي.

لنبين خطوات خوارزمية الطفرة على مستوى الجيل والصبغي:

- على مستوى الجيل:
  - توليد عدد عشوائي حقيقي ضمن المجال [0,1].
  - إذا كان العدد أكبر من احتمال الطفرة (Pm) انتقل للخطوة الأخيرة.
  - إذا كان العدد أصغر أو يساوي احتمال الطفرة (Pm)، فقم بتوليد عدد عشوائي بين الواحد وحجم الجيل، أي ضمن المجال [1,Population Size].

- قم باختيار الصبغي ذي الترتيب الموافق للعدد العشوائي المولد في الخطوة السابقة.
- قم بتوليد عددين عشوائيين صحيحين مختلفين بين الواحد وطول الصبغي، أي ضمن المجال [1,Chromosome Length].
- قم بمبادلة الصبغيين المحددين في الخطوة السابقة.
- انتهاء الخوارزمية.

مثال:

لدينا جيل من الصبغيات المرمزة عشرياً بحجم (Ps=8)، نفذت على الجيل عملية الانتخاب ومن ثم العبور فحصلنا على الجيل المبين. المطلوب، أوجد جيل الأبناء بعد تنفيذ مرحلة الطفرة على مستوى الجيل، علماً أن (Pm=0.25).

$$\text{Pop} = \begin{bmatrix} 2 & 9 & 6 & 5 & 1 & 7 & 8 & 3 & 4 \\ 5 & 3 & 4 & 6 & 9 & 1 & 2 & 8 & 7 \\ 9 & 8 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 1 & 3 & 2 & 9 & 8 & 5 & 6 & 4 & 7 \\ 9 & 3 & 2 & 8 & 1 & 5 & 7 & 4 & 6 \\ 4 & 7 & 3 & 6 & 5 & 1 & 8 & 2 & 9 \\ 7 & 9 & 8 & 4 & 5 & 2 & 6 & 1 & 3 \\ 6 & 1 & 4 & 3 & 9 & 5 & 7 & 8 & 2 \end{bmatrix}$$

يبدأ الحل بتوليد عدد عشوائي ضمن المجال [0,1]، وليكن العدد (0.14). بما أن العدد أصغر من احتمال الطفرة (Pm)، نقوم بتوليد عدد عشوائي بين الواحد وحجم الجيل، أي ضمن المجال [1,10].

ليكن العدد العشوائي المولد هو (5). نختار الصبغي ذي الترتيب الموافق للعدد العشوائي المولد، أي الصبغي الخامس في مصفوفة الجيل.

نقوم بعد اختيار الصبغي الواجب تنفيذ الطفرة عليه بتوليد عددين عشوائي صحيحين مختلفين بين الواحد وطول الصبغي، أي ضمن المجال [1,9].

ليكن العددان العشوائيان المولدان هما (2,6). نختار المورثة الثانية والسادسة في الصبغي الخامس من مصفوفة الجيل ونبادل بينهما، ليكون الجيل الجديد أي جيل الأبناء كما هو مبين.

$$\text{Pop} = \begin{bmatrix} 2 & 9 & 6 & 5 & 1 & 7 & 8 & 3 & 4 \\ 5 & 3 & 4 & 6 & 9 & 1 & 2 & 8 & 7 \\ 9 & 8 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 1 & 3 & 2 & 9 & 8 & 5 & 6 & 4 & 7 \\ 9 & \boxed{5} & 2 & 8 & 1 & \boxed{3} & 7 & 4 & 6 \\ 4 & 7 & 3 & 6 & 5 & 1 & 8 & 2 & 9 \\ 7 & 9 & 8 & 4 & 5 & 2 & 6 & 1 & 3 \\ 6 & 1 & 4 & 3 & 9 & 5 & 7 & 8 & 2 \end{bmatrix}$$

- على مستوى الصبغي:
  - تكرار الخطوات اللاحقة على عدد الصبغيات أي بحجم الجيل، وعند الانتهاء الانتقال للخطوة الأخيرة.
  - توليد عدد عشوائي حقيقي ضمن المجال [0,1].
  - إذا كان العدد أكبر من احتمال الطفرة (Pm) انتقل للخطوة الأولى.

- إذا كان العدد أصغر أو يساوي احتمال الطفرة (Pm)، فقم بتوليد عددين عشوائيين صحيحين مختلفين بين الواحد وطول الصبغي، أي ضمن المجال [1,Chromosome Length].
- قم بمبادلة الصبغيين المحددين في الخطوة السابقة.
- العودة للخطوة الأولى.
- انتهاء الخوارزمية.

مثال:

لدينا جيل من الصبغيات المرمزة عشرياً بحجم (Ps=8)، نفذت على الجيل عملية الانتخاب ومن ثم العبور فحصلنا على الجيل المبين. المطلوب، أوجد جيل الأبناء بعد تنفيذ مرحلة الطفرة على مستوى الصبغي، علماً أن (Pm=0.07).

$$\text{Pop} = \begin{bmatrix} 2 & 9 & 6 & 5 & 1 & 7 & 8 & 3 & 4 \\ 5 & 3 & 4 & 6 & 9 & 1 & 2 & 8 & 7 \\ 9 & 8 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 1 & 3 & 2 & 9 & 8 & 5 & 6 & 4 & 7 \\ 9 & 3 & 2 & 8 & 1 & 5 & 7 & 4 & 6 \\ 4 & 7 & 3 & 6 & 5 & 1 & 8 & 2 & 9 \\ 7 & 9 & 8 & 4 & 5 & 2 & 6 & 1 & 3 \\ 6 & 1 & 4 & 3 & 9 & 5 & 7 & 8 & 2 \end{bmatrix}$$

بشكل مشابه لحالة الطفرة على مستوى الصبغي في الترميز الثنائي، وبافتراض أن الخوارزمية نفذت وكانت الأعداد العشوائية المولدة بالشكل الآتي:

RN=[0.32 0.05 0.89 0.34

0.17 0.29 0.98 0.64]

تحدث الطفرة في الصبغي الثاني فقط، لأن العدد العشوائي المولد أصغر أو يساوي احتمال الطفرة في الدليل الثاني ضمن مصفوفة (RN).

نبين فيما يأتي الجيل الجديد أي جيل الأبناء بعد إجراء الطفرة على الصبغي الثاني، وبمبادلة المورثتين الأولى والرابعة المحددتين عشوائياً، كما سبق وبينا سابقاً.

$$\mathrm{Pop} = \begin{bmatrix} 2 & 9 & 6 & 5 & 1 & 7 & 8 & 3 & 4 \\ \boxed{6} & 3 & 4 & \boxed{5} & 9 & 1 & 2 & 8 & 7 \\ 9 & 8 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 1 & 3 & 2 & 9 & 8 & 5 & 6 & 4 & 7 \\ 9 & 3 & 2 & 8 & 1 & 5 & 7 & 4 & 6 \\ 4 & 7 & 3 & 6 & 5 & 1 & 8 & 2 & 9 \\ 7 & 9 & 8 & 4 & 5 & 2 & 6 & 1 & 3 \\ 6 & 1 & 4 & 3 & 9 & 5 & 7 & 8 & 2 \end{bmatrix}$$

تتوافر عدة طرائق أخرى للقيام بعملية الطفرة في الترميز العشري، ومنها طريقة الخلط (Scramble Mutation). في طريقة الخلط يتم اختيار مجموعة عشوائية متتابعة من المورثات ومن ثم خلطهم عشوائياً وإعادتهم بتوزيع جديد إلى الصبغي. يبين الشكل(6-33) كيفية حدوث الطفرة في صبغي مرمز عشرياً باستخدام طريقة الخلط، حيث تم اختيار بداية ونهاية المجموعة عشوائياً، بعد ذلك تم خلطهم عشوائياً أيضاً.

قبل الطفرة	Chromosome A	3	**5**	**2**	**6**	**1**	7	4
		↓	↔	↔	↔	↔	↓	↓
بعد الطفرة	Chromosome A'	3	**6**	**5**	**1**	**2**	7	4

الشكل(6-33) حدوث طفرة على صبغي مرمز عشرياً باستخدام طريقة الخلط

3) الطفرة في ترميز القيمة (Value Encoding Mutation):

يوجد أنواع كثيرة من الطرائق المستخدمة في ترميز الصبغيات بالقيمة، لكن سوف نركز على ترميز القيمة الذي يستخدم الأعداد الحقيقية. تكمن فكرة الطفرة في هذا النوع من الترميز بتغيير قيمة المورثة بمقدار صغير. يحدد هذا المقدار من قبل المستخدم في مرحلة التأهيل الابتدائي، إما كقيمة جبرية ثابتة، أو كنسبة مئوية من قيمة المورثة. يمكن اختيار تغيير قيمة مورثة واحدة أو مورثتين، الأولى بقيمة موجبة والثانية بقيمة سالبة. تختار المورثات التي يجري عليها التغيير بشكل عشوائي.

يبين الشكل(6-34) كيفية حدوث طفرة في صبغي تم ترميزه بالقيمة باستخدام أعداد حقيقية، حيث تم اختيار التغيير بقيمة جبرية ثابتة مقدارها (0.3)، وتغيير مورثتين عشوائيتين بقيم متعاكسة. تم إضافة (0.3) للمورثة الثانية وطرح القيمة نفسها من المورثة الخامسة.

قبل الطفرة	Chromosome A	2.3	**7.0**	4.4	2.6	**1.8**
		↓	+	↓	↓	-
بعد الطفرة	Chromosome A'	2.3	**7.3**	4.4	2.6	**1.5**

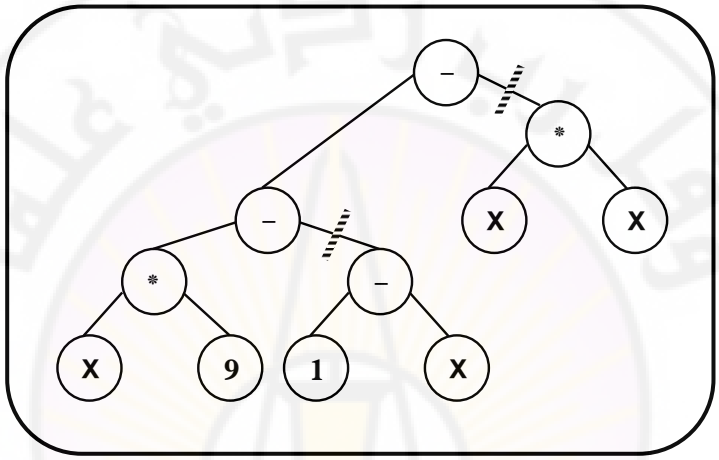
الشكل(6-34) حدوث طفرة على صبغي مرمز بالقيمة

بما أن ترميز القيمة هو الترميز الأكثر عمومية، فإن طريقة الطفرة تختلف حسب الرموز والطريقة المستخدمة في الترميز. فعند حل مسألة إيجاد مسار إنسالة (Robot) وفق شروط محددة، فمن الممكن أن يكون الصبغي عبارة عن مجموعة متتالية مسموحة من اتجاهات التحرك. في هذه الحالة يمكن تنفيذ الطفرة بما يشبه الطريقة المستخدمة في الترميز العشري، أي بتبديل قيم مورثتين عشوائيتين ضمن الصبغي.

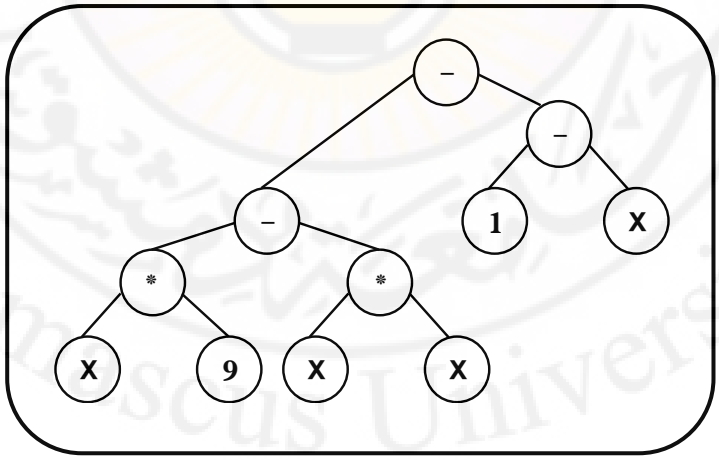
4) الطفرة في الترميز الشجري (Tree Encoding Mutation):

كما في ترميز القيمة، فإن طريقة حل المسألة تفرض الطريقة الأنسب في إجراء الطفرة في الترميز الشجري. فمن الممكن أن تتم عن طريق تغيير مُحَدِدٍ أو عدد في الشجرة، وقد يتم التبديل بين فرعين من الشجرة ابتداءً من نقاط الارتباط المختارة عشوائياً من الشجرة . يترك للمستخدم اختيار الطريقة التي تناسبه، مع تحديد محددات الطريقة في مرحلة التأهيل الابتدائي.

يبين الشكل(6-35) إحدى الصبغيات المرمزة شجرياً قبل إجراء الطفرة، مع تحديد أماكن قص الأفرع المطلوب مبادلتها. أما الشكل(6-36) فيبين الصبغي بعد إجراء عملية الطفرة، حيث نلاحظ كيف تمت مبادلة فرعي الشجرة.



الشكل(6-35) صبغي مرمز شجرياً قبل الطفرة



الشكل(6-36) صبغي مرمز شجرياً بعد الطفرة

6-15- البرمجة الجينية ((Genetic Programming (GP)):

6-15-1- مقدمة:

إن الهدف الأساسي من الذكاء الصنعي والتعلم الآلي، هو الاستفادة من علم الحاسوب للحصول على نظام آلي، يمكننا من حل المسائل بشكل يبدو وكأنه ناتج عن حل بشري ذكي.

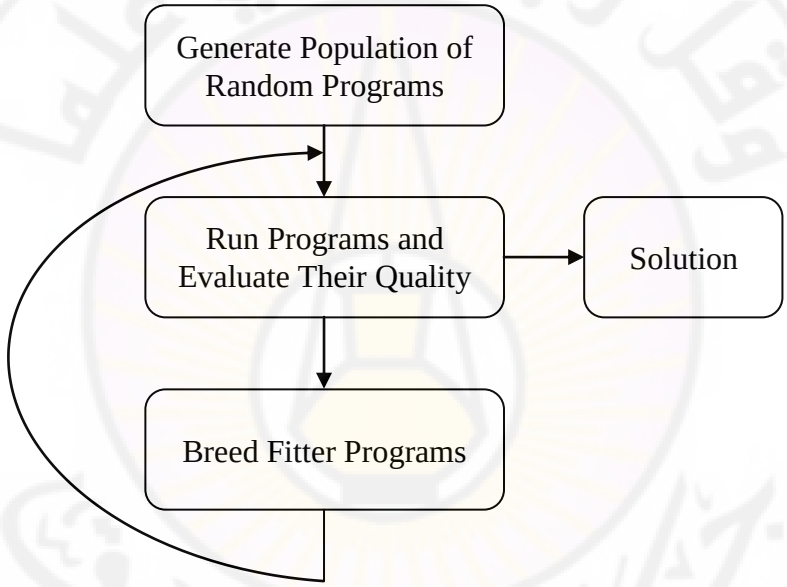
تعد البرمجة الجينية التي انطلقت على يد (John R. Koza) في عام (1992)، إحدى تقنيات الحوسبة التطورية (Evolutionary Computation)، التي يمكنها حل المسائل دون الحاجة لكثير من المعطيات والمحددات، ودون تحديد خطوات الحل أو معرفة الخوارزمية التدفقية اللازم إتباعها للوصول للحل. يمكن القول إن البرمجة الجينية طريقة متقدمة في حل المسائل المعقدة انطلاقاً من عبارات وأوامر عالية المستوى باستخدام الحواسيب. وقد ازداد الاهتمام بالبرمجة الجينية في السنوات الأخيرة، نتيجة التطور العلمي وازدياد تعقيد النظم البرمجية والحاجة لحل مسائل في غاية الصعوبة، كما كان لاهتمام الباحثين والمطورين ونشرهم للكثير من الأبحاث والمقالات دور بارز في نشر هذا الفرع من علم الخوارزميات الجينية.

البرمجة الجينية حالة خاصة من الخوارزميات التطورية (Evolutionary Algorithm)، حيث تكون الصبغيات عبارة عن برامج حاسوبية تجري عليها كافة مراحل الخوارزميات الجينية من انتخاب وعبور وطفرة للوصول إلى البرنامج الأمثل.

يوضح الشكل(6-37) النواة البرمجية للبرمجة الجينية. تبدأ العملية بتوليد جيل ابتدائي عشوائي من البرامج, بعد ذلك تبدأ حلقة تكرارية تختبر فيها ملاءمة كل برنامج، وتنتخب بعض البرامج وفق خوارزميات الانتخاب المعروفة وتنفذ

عملية العبور والطفرة للحصول على جيل جديد من البرامج. تتوقف الحلقة عند تحقق شرط التوقف كالوصول إلى درجة ملاءمة محددة لأحد البرامج.

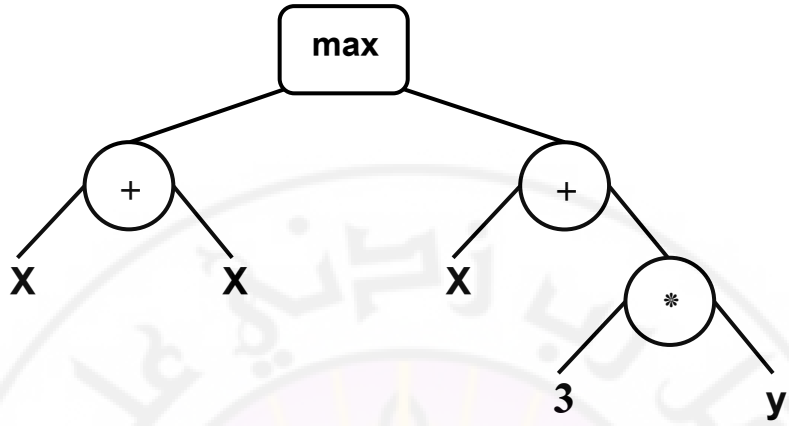
تدعم بعض أنظمة البرمجة الجينية الحلول البنيوية، التي تسمح بإجراء تغيير على هيكلية البرنامج، كعدد التوابع والإجراءات ضمن البرنامج، وعدد فروع الشجرة الممثلة للبرنامج.



الشكل(6-37) النواة البرمجية للبرمجة الجينية

6-15-2- مقدمة الترميز والتهيئة الابتدائية في (GP):

يعبر عن البرنامج في البرمجة الجينية كأشجار، عوضاً عن استخدام التعليمات البرمجية المكتوبة بأسطر متتالية. يوضح الشكل(6-38) تمثيلاً شجرياً لبرنامج القيمة العظمى (max(x*x, x+3y)).



الشكل(6-38) التمثيل الشجري لبرنامج القيمة العظمى

إن المتغيرات والثوابت في البرنامج (x, y, 3)، تدعى بالمحطات الطرفية (Terminals) في (GP)، وتمثل بأوراق الشجرة. بينما العمليات الرياضية (+ , * , max)، فتدعى بالتوابع (Functions) في (GP)، وتمثل بعقد (internal Node) الشجرة. وبالتالي تكون المجموعة الابتدائية (Primitive Set) في (GP) هي مجموعة التوابع والطرفيات.

تبين الجداول الآتية بعض الأمثلة عن مجموعة التوابع ومجموعة المحطات الطرفية.

Function Set	
Kind of Primitive	Examples
Arithmetic	+, −, *, /, ^
Mathematical	Sin, Cos, Exp
Boolean	AND, OR, NOT
Conditional	IF – THEN – ELSE
Looping	FOR, REPEAT

Terminal Set	
Kind of Primitive	Examples
Variables	x, y, z
Constant Values	6, 43, 0.56
Variant Functions	rand, go left

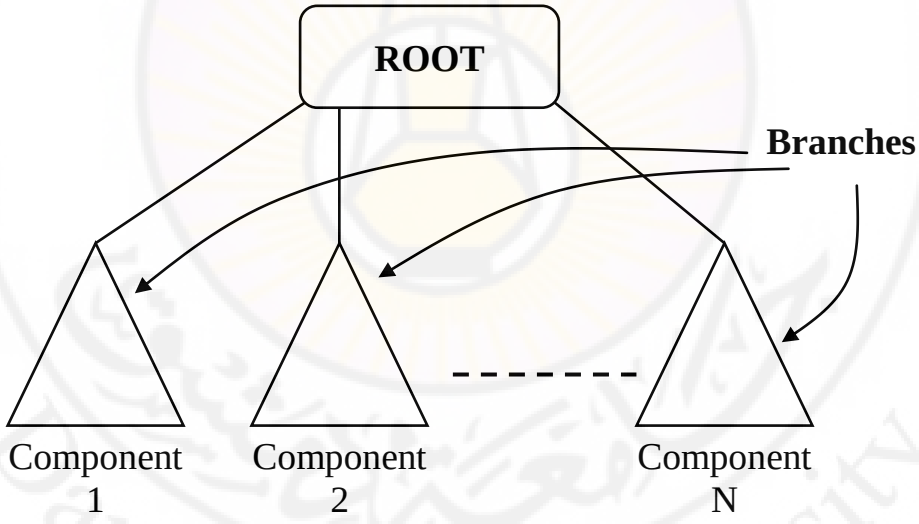
عادة، تكون مجموعة التوابع المستخدمة في (GP) متعلقة بطبيعة المسألة، فمن الممكن استخدام العمليات الحسابية فقط عندما تكون المسألة بسيطة. يجب الانتباه عند استخدام بعض الخيارات من مجموعة المحطات الطرفية كمولد الأعداد العشوائية (rand)، يسبب ذلك تبايناً في سلوك البرنامج، لأنه عند كل تنفيذ للبرنامج سوف يقوم مولد الأعداد العشوائية بتوليد قيم مختلفة، وبالتالي ستكون النتائج مختلفة أيضاً، ولو كانت معطيات الدخل ثابتة ومتطابقة عند كل تنفيذ للبرنامج. فالبرنامج الأمثل الذي ينتج عن تنفيذ برنامج خوارزمية البرمجة الجينية لن يكون متطابقاً عند كل تنفيذ، وهذا غير مقبول من الناحية التطبيقية. يكمن الحل عند الحاجة لاستخدام (rand)، في الاحتفاظ بالقيم العشوائية عند الوصول للحل الأمثل، لجعلها قيماً ثابتة في نسخة البرنامج النهائي.

يجب أن تتصف معظم مجموعة التوابع بخاصية هامة تدعى الإغلاق (Closure). تضمن هذه الخاصية سلامة وصحة البرامج. فعندما تكون العقدة تابعاً مثل (IF)، فهذا يتطلب وجود شرط منطقي، عند تحققه تنفذ كتلة برمجية أولى، وعند عدم تحققه تنفذ كتلة ثانية. بالطبع، ليس من المعقول إجراء طفرة مثلاً على هذه العقدة تؤدي لتغييرها من (IF) إلى (+). كما أن عمليات العبور سوف تؤدي إلى قطع فرع من الشجرة واستبداله، فهل سيضمن هذا الاستبدال المحافظة

على صحة البرنامج ؟. تضمن خاصية الإغلاق تجاوز العقبات الممكن مصادفتها خلال العمليات التطورية للبرنامج.

تتكون البرامج في البرمجة الجينية المتقدمة من عناصر أو مكونات (Components) متعددة، كالبرامج الفرعية (Subroutines)، وفي هذه الحالة يكون التمثيل المستخدم في (GP) عبارة عن مجموعة من الأشجار الفرعية المجمعة تحت عقدة الجذر (ROOT)، والتي تدعى بالأغصان أو الفروع (Branches)، وهذا ما يوضحه الشكل(6-39).

يشكل عدد ونوع الفروع مع مواصفات أخرى محددة للفروع، البنية الهيكلية للبرنامج.



الشكل(6-39) تمثيل برنامج متعدد المكونات

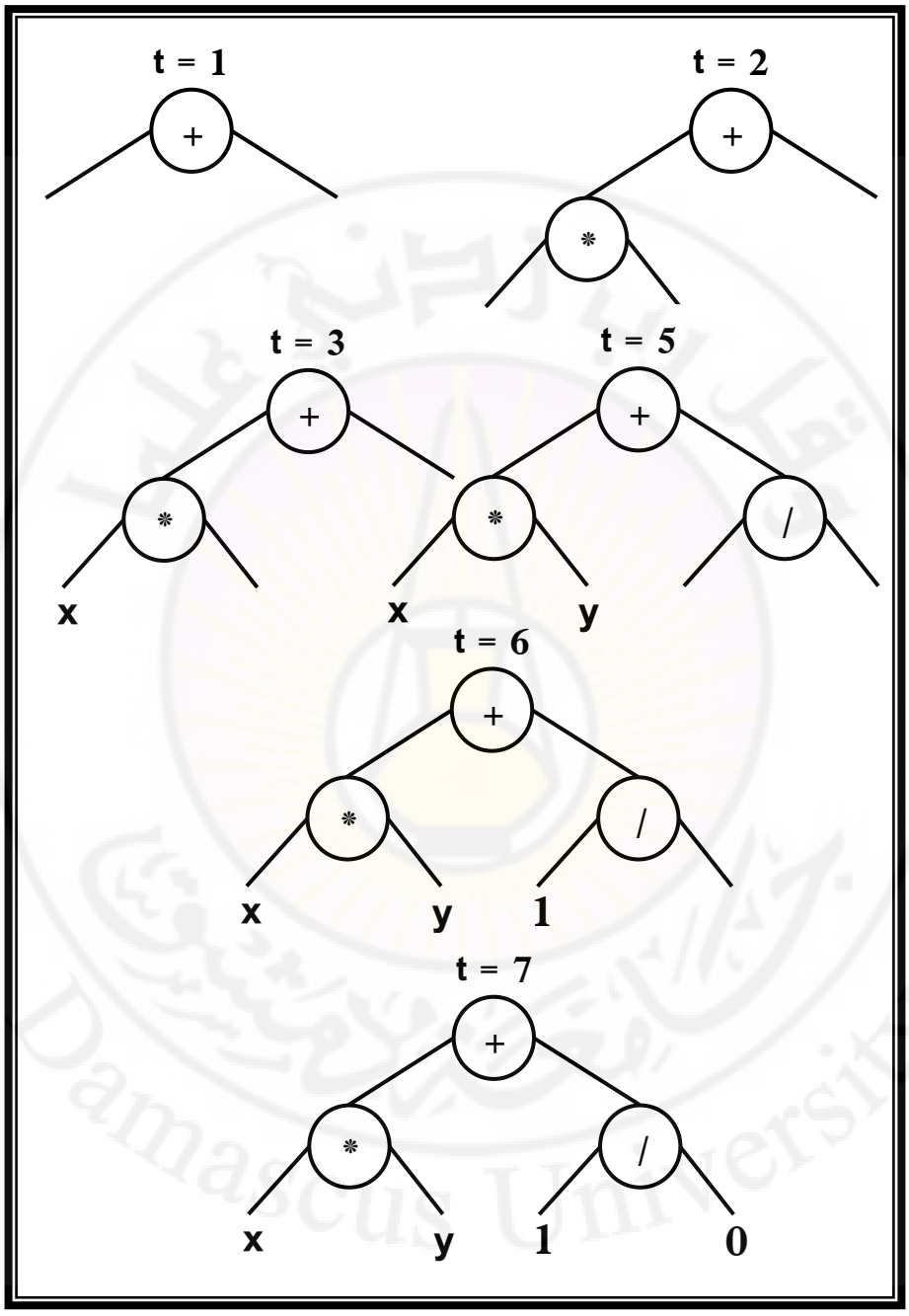
من الشائع في (GP) استخدام أساليب التعبير المستخدمة في بعض لغات الذكاء الصنعي مثل (LISP)، فمثلاً (max(x*x, x+3y))، تكتب بالشكل ((max(* x x)(+ x (* 3 y)))). يسهل هذا الأسلوب في التعبير (Prefix) معرفة العلاقة بين مكونات الشجرة كافة.

يعتمد تنفيذ الأشجار في (GP) بشكل كبير على لغة البرمجة والمكتبات المستخدمة. إن معظم لغات الذكاء الصنعي (LISP, Prolog)، واللغات الحديثة (Python, Ruby, C#)، والبيئات البرمجية (MATLAB, Mathematica)، تدعم أنواع البيانات الأساسية التي تجعل من السهل تنفيذ الأشجار كالقوائم الديناميكية وجامع البيانات المهملة (Automatic Garbage Collector) ذو الأهمية الكبيرة في إدارة وتنظيم الذاكرة. بينما لا يتوافر هذا الدعم في لغات أخرى (C, C++,)، ما يتطلب تنفيذاً برمجياً من قبل المبرمج أو استخدام مكتبات داعمة لذلك.

يتم توليد الجيل الابتدائي عشوائياً بعدة طرائق؛ منها:

1) الطريقة الكاملة (Full Method):

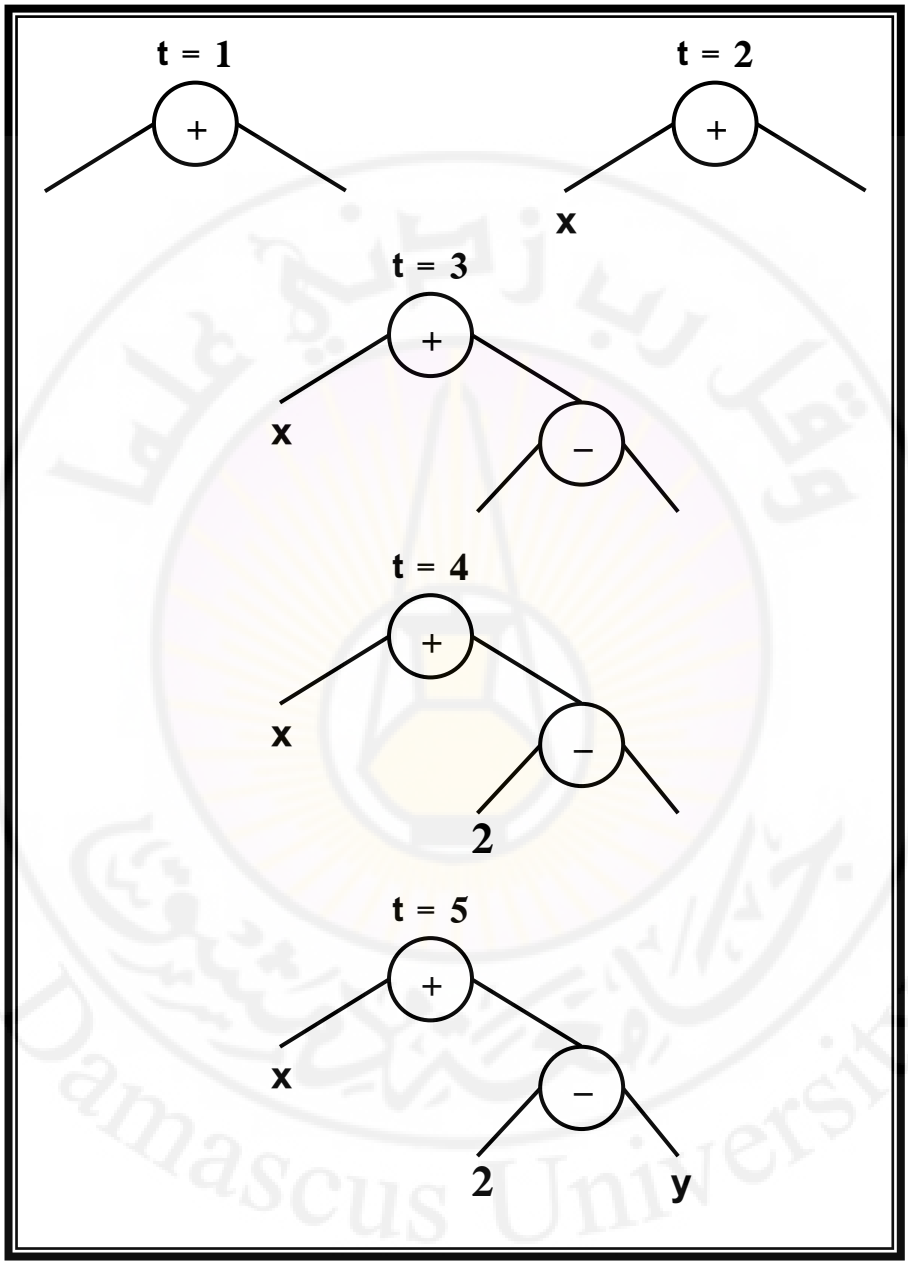
يتم توليد أشجار كاملة بالتقاط العقد عشوائياً من مجموعة التوابع حتى الوصول إلى أقصى عمق الشجرة، ولا يمكن اختيار الطرفيات إلا بعد ذلك العمق. يبين الشكل(6-40) خطوات بناء شجرة كاملة بعمق (2)، وبالتالي تتكون من (7) عقد. نلاحظ أن الطريقة الكاملة تولد أشجاراً ذات حجم وشكل محددين.



الشكل(6-40) خطوات بناء شجرة كاملة بعمق (2)

2) طريقة النمو (Grow Method):

يتم توليد الأشجار بالتقاط العقد عشوائياً من المجموعة الابتدائية المتضمنة التوابع والطرفيات حتى الوصول إلى أقصى عمق الشجرة، وعندها لا يمكن اختيار إلا الطرفيات. يبين الشكل(6-41) خطوات بناء شجرة بعمق (2) بطريقة النمو. نلاحظ أن طريقة النمو تولد أشجاراً ذات أحجام وأشكال متنوعة.



الشكل(6–41) خطوات بناء شجرة بعمق (2) بطريقة النمو

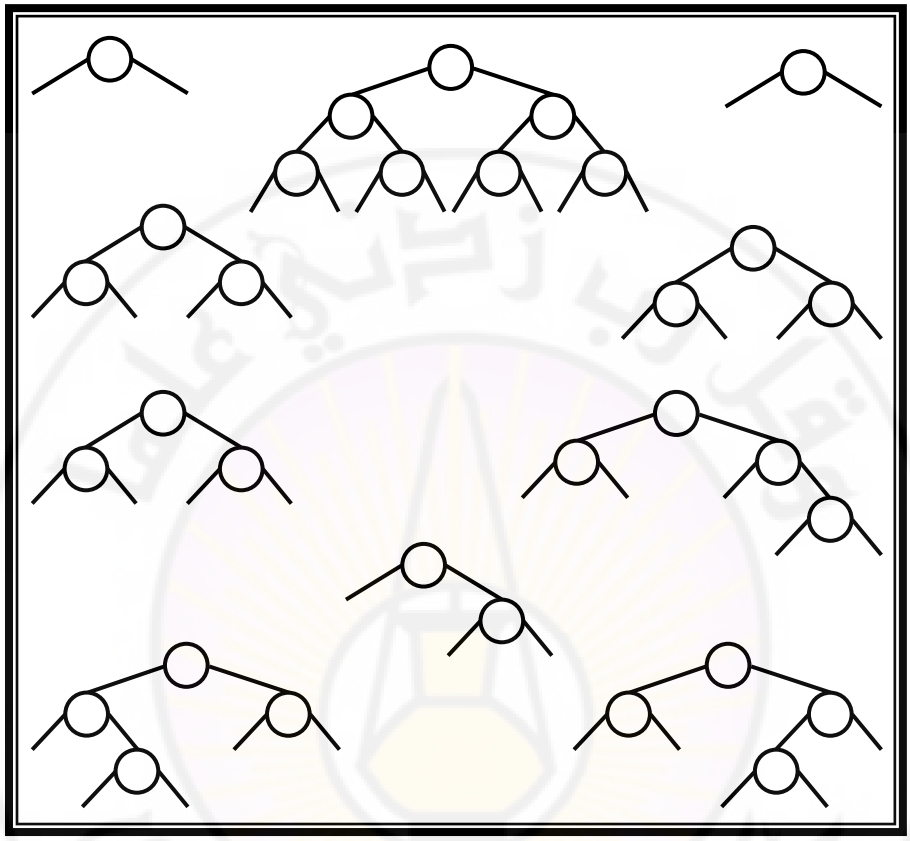
3) طريقة الانحدار (Ramped half-and-half Method):

من الملاحظ في طريقة النمو تعلق حجم وشكل الأشجار بحجم مجموعة التوابع والطرفيات. فعندما يكون عدد الطرفيات الممكن اختيارها أكبر من عدد التوابع المتاحة، فإنه غالباً ما ينتج عن طريقة النمو شجرة قصيرة لأن احتمال وضع طرفية أكبر من احتمال وضع تابع في العقدة، وبما أن أوراق الشجرة تكون فقط طرفيات، فإنها ستحد من تطور عمق الشجرة للحد الأقصى. أما عندما يكون عدد التوابع الممكن اختيارها أكبر من عدد الطرفيات المتاحة، فإنه غالباً ما ينتج عن طريقة النمو شجرة طويلة مشابهة للطريقة الكاملة لأن احتمال وضع تابع أكبر من احتمال وضع طرفية في العقدة، حتى الوصول للحد الأقصى لعمق الشجرة.

للجمع بين كلتا الطريقتين، فقد اقترح (Koza) طريقة الانحدار، التي تعتمد على إنشاء نصف الجيل الابتدائي باستخدام الطريقة الكاملة، والنصف الآخر باستخدام طريقة النمو. كما يتم اعتماد مجال لقيم العمق، ما يضمن التنوع في أحجام وأشكال الأشجار، وهذا هو سبب التسمية (Ramped).

يبين الشكل(6-42) جيلاً بحجم (10) صبغيات مبنياً بطريقة (Ramped half-and-half) وبمجال عمق [1,3]. نلاحظ أن الأشجار ذات أحجام وأشكال متنوعة، مع إمكانية بناء شجرة كاملة بطريقة النمو.

بالطبع، ليس من الضروري أن يكون الجيل الابتدائي عشوائياً بكل معنى الكلمة. فإذا كانت بعض خصائص الحل معروفة، فمن الممكن الاستفادة من ذلك، لإنشاء أشجار تتمتع بالخصائص نفسها، ما يجعل الوصول للحل الأمثل أسرع. فنقطة الانطلاق من الجيل الابتدائي لها أهمية كبيرة في توجيه الخوارزمية نحو الحل الأمثل. الأشجار المنشأة كنقطة بداية في الجيل الابتدائي قد تنتج عن تدخل بشري مباشر، أو نتيجة تنفيذ برنامج محدد.



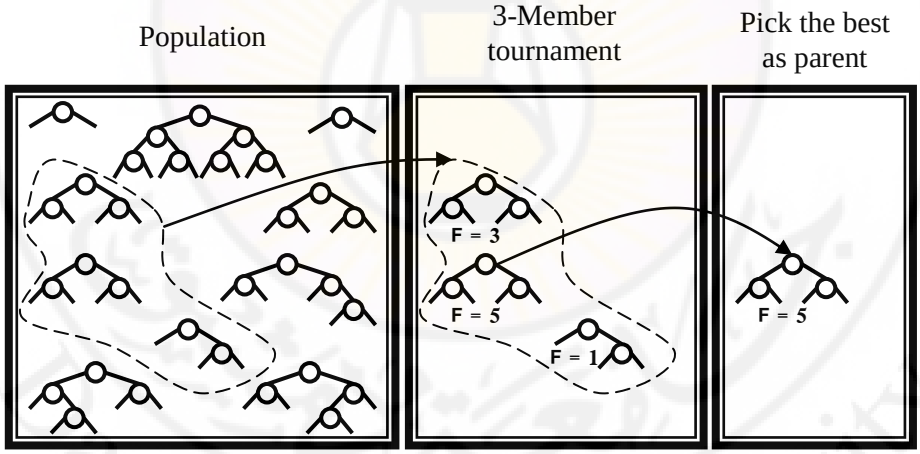
الشكل(6-42) جيلاً مبنياً بطريقة (Ramped)

6-15-3- الانتخاب في (GP):

تطبق مرحلة الانتخاب في البرمجة الجينية بشكل مشابه للطريقة المتبعة في الخوارزميات الجينية، حيث يتم اختيار الصبغيات وفق محدد احتمالي يأخذ بالاعتبار قيم الملاءمة للصبغيات. يكون احتمال اختيار الصبغي الأكثر ملاءمة أعلى من غيره، وبذلك يتوقع أن ينتج الجيل الجديد من أفضل الصبغيات في الجيل القديم، ما يزيد فرصة الوصول للحل الأمثل بشكل أسرع.

من أكثر خوارزميات الانتخاب استخداماً في البرمجة الجينية هي خوارزمية الانتخاب على مبدأ مسابقة (n) عضو (n–Member Tournament Selection)، حيث ينتخب (n) صبغي من الجيل الحالي بشكل عشوائي، ثم تجرى مسابقة بينهم حسب مقياس الملاءمة، لينتخب الصبغي الذي يملك مقياس ملاءمة أعلى من غيره. تكرر العملية السابقة حتى يستكمل الجيل، أي عدداً من المرات يساوي حجم الجيل . تحدد قيمة (n) في مرحلة التأهيل الابتدائي من قبل المستخدم.

يوضح الشكل(6–43) كيفية تنفيذ خطوة انتخاب واحدة في جيل من الصبغيات المرمزة شجرياً في البرمجة الجينية، مع اختيار (3) أعضاء للمسابقة.



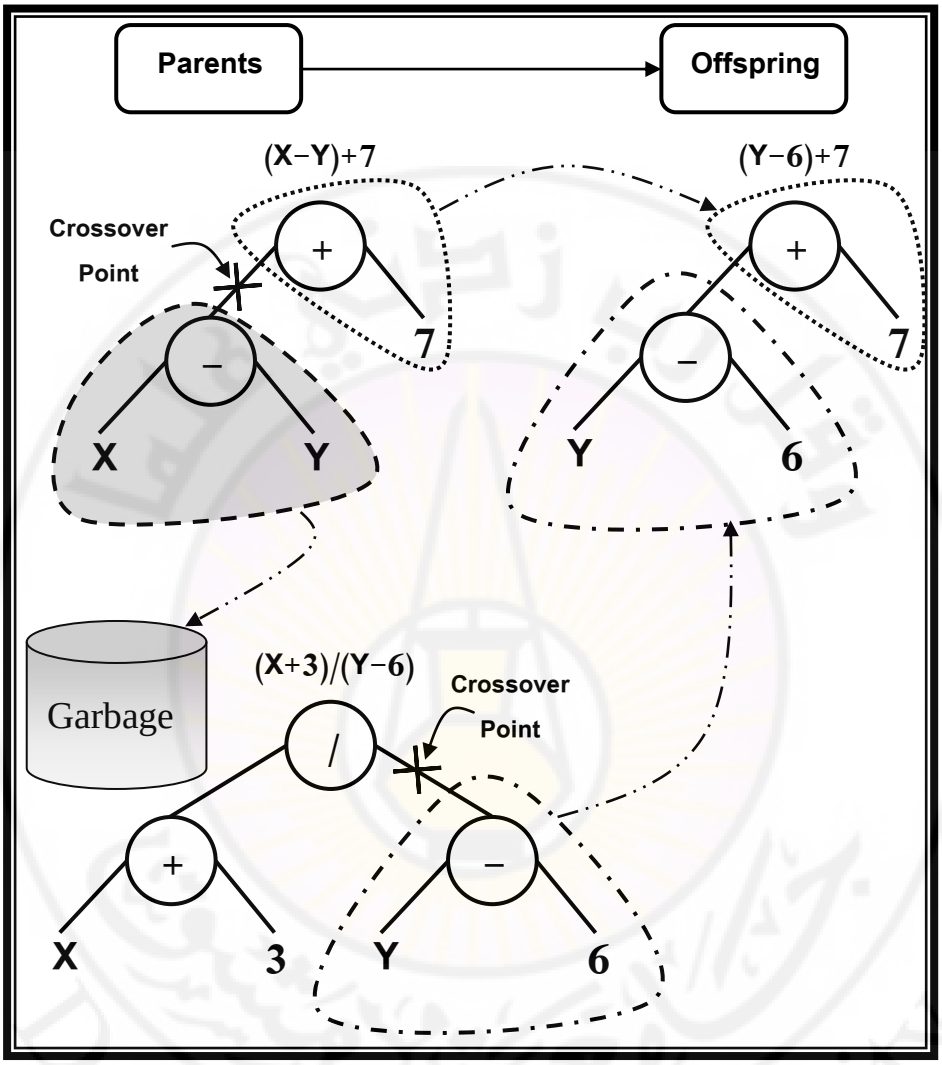
الشكل(6–43) انتخاب الصبغيات المرمزة شجرياً في البرمجة الجينية

6–15–4– العبور في (GP):

ينفذ العبور في البرمجة الجينية بشكل أكثر صعوبة مما هو مستخدم في الخوارزميات الجينية. من أبسط الطرائق المستخدمة هي تحديد نقاط عبور في

شجرتي الوالدين وإجراء عملية تبادل للشجرة الفرعية بين كلا الوالدين للحصول على الذرية الجديدة من الأبناء. يوضح الشكل(6-44) تنفيذ عملية العبور بين والدين للحصول على ابن واحد. يراعى في جميع الطرائق المستخدمة في عملية العبور المحافظة على بنية الشجرة بحيث لا يزيد العمق أو تتغير البنية، وهذا يتطلب بعض القيود في اختيار نقاط العبور، كأن تكون بالسوية نفسها وبالتالي تتشابه بنية الأشجار الفرعية.

بينت الأبحاث المتعلقة بعملية العبور في البرمجة الجينية أنه من الأفضل اختيار نقاط العبور باحتمال متغير، وذلك لأنه عندما يزداد العمق تصبح غالبية العقد محطات طرفية أي أوراقاً في الشجرة، وهذا يعني أن غالبية عمليات العبور لن تحتوي على مورثات كثيرة، وستكون مجرد مبادلة ورقتين فقط. كحل مقبول لهذه المشكلة فقد اقترح (Koza) أن يتم اختيار التوابع باحتمال (0.9)، واختيار الأوراق باحتمال (0.1) مثلاً، وبذلك تكون فرص مبادلة التوابع والأشجار الفرعية قريبة من فرص مبادلة أوراق الشجرة فقط.



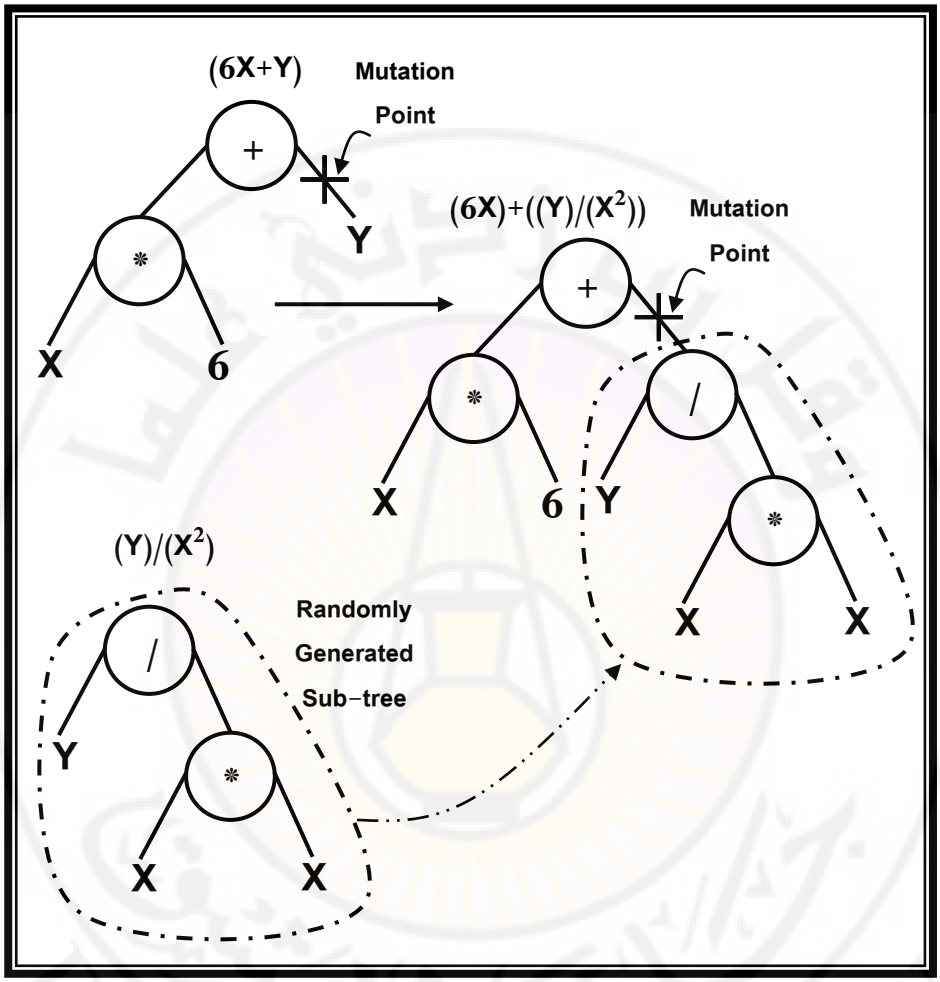
الشكل(6-44) عملية العبور في البرمجة الجينية

6-15-5- الطفرة في (GP):

لم تكن الطفرة موجودة في بدايات البرمجة الجينية، إلا أنها حالياً مستخدمة على نطاق واسع، وبطرائق مختلفة نبين بعضها بشكل سريع.

من أكثر الخوارزميات استخداماً للقيام بعملية الطفرة هنا هي الشجرة الفرعية (subtree mutation)، التي تعتمد على اختيار نقطة قطع في الصبغي المعبر عنه كشجرة متعددة الفروع، ومن ثم استبدال شجرة عشوائية البناء، بالشجرة الفرعية من نقطة القطع في الشجرة الأصلية. تتشكل الشجرة العشوائية بشكل لحظي عند إقرار عملية الاستبدال. تجري هذه العملية وفق قيود تحددها المسألة المدروسة، فبناء الشجرة الفرعية يخضع لقيود محددة، وليس عشوائياً بكل ما تعنيه الكلمة. يمكن تنفيذ خوارزمية (subtree mutation) بإجراء عملية عبور بين البرنامج الممثل للشجرة وبين برنامج جديد يبنى عشوائياً، وهي عملية معروفة باسم الدجاج مقطوع الرأس (Headless Chicken). يبين الشكل(6-45) تنفيذ عملية الطفرة وفق طريقة (subtree mutation).

يستخدم أحياناً أسلوباً أقل كلفة زمنية وبرمجية وهو (point mutation)، الذي يعتمد على اختيار عقدة عشوائياً من الشجرة، وإجراء تبديل في قيمتها من مجموعة القيم المتاحة. فإذا كانت العقدة تابعاً فيجب اختيار تابع جديد من مجموعة التوابع المتوافرة لحل المسألة. وإذا كانت العقدة ورقة فيجب اختيار ورقة جديدة من مجموعة المحطات الطرفية المتوافرة لحل المسألة.

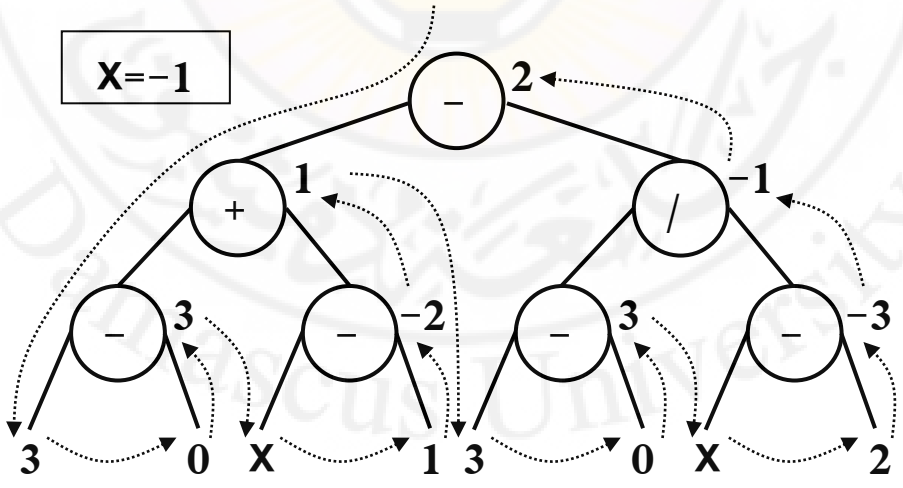


الشكل(6-45) الطفرة وفق طريقة (subtree mutation)

6-15-6- تابع الملاءمة في (GP):

تحدد المجموعة الابتدائية فضاء البحث، فكلما ازداد حجم المجموعة، كلما توسع فضاء البحث الذي يتطلب اختبار جميع الاحتمالات المتوافرة لإنشاء البرامج

بجميع خيارات المجموعة الابتدائية. من ناحية أولى، فإنه لا يمكن اختبار جميع الخيارات المتاحة في فضاء البحث، بسبب العدد الهائل الذي يتطلب كلفة زمنية كبيرة تصل أحياناً لسنوات عديدة على أسرع الحواسيب. ومن ناحية ثانية، فإنه لا يوجد مؤشر يدل على المنطقة التي يتواجد فيها الحل الأمثل ضمن فضاء البحث عموماً. هذه الأسباب تتطلب إيجاد مؤشر يدل على مدى الاقتراب من الحل الأمثل، أو على أقل تقدير، الحصول على حل مقبول يحقق القيود المرغوبة. كما في الخوارزميات الجينية فإن تابع الملاءمة هو المؤشر المستخدم للدلالة على دقة الحل، والتي تحدد في (GP) بمقدار الفرق بين خرج البرنامج والخرج المرغوب. إن كون الحل أو الصبغي في (GP) عبارة عن ترميز برمجي، فهذا يتطلب القيام بتنفيذ البرنامج المختبر عدة مرات لتحديد قيمة تابع الملاءمة ، وعدم الاكتفاء بمرة واحدة. يتطلب تنفيذ البرنامج القيام بحساب قيمة كل عقدة وفق ترتيب يضمن معرفة قيم وسائط العقدة قبل حساب العقدة. يبين الشكل(6-46) عملية إيجاد قيمة شجرة بسيطة مع اعتبار المتحول (X=-1).



الشكل(6-46) إيجاد قيمة شجرة بسيطة

6-15-7- مثال في (GP):

لتكن لدينا مجموعة من النقاط المعطاة بإحداثياتها الديكارتية. يطلب إيجاد التابع الصحيح الممثل لمجموعة النقاط، باستخدام البرمجة الجينية.

x	−2	−1.5	−1	−0.5	0	0.5	1	1.5	2
y	3	1.7	1	0.7	1	1.7	3	4.7	7

يبدأ الحل بتعريف مجموعة الطرفيات (T). باختيار تمثيل النقاط بكثير حدود أو بتابع صحيح، فهذا يتطلب تعريف متحول (x) لتمثيل التابع (y)، كما هو مبين في الجدول السابق. إن أمثال المتحول (x) هي أعداد عشوائية تنتمي إلى مجموعة الأعداد الحقيقية، ولنأخذ المجال [5.0, 5.0−]، وبالتالي تصبح مجموعة الطرفيات ($T=\{x, \Re\}$).

أما مجموعة التوابع (F) فسوف نأخذ فقط العمليات الحسابية الأساسية وهي عمليات الجمع والطرح والضرب والقسمة، لأن المطلوب تابع صحيح، وبالتالي تصبح مجموعة التوابع ($F=\{+, -, *, /\}$)، مع الحرص على منع وجود عملية تقسيم على الصفر.

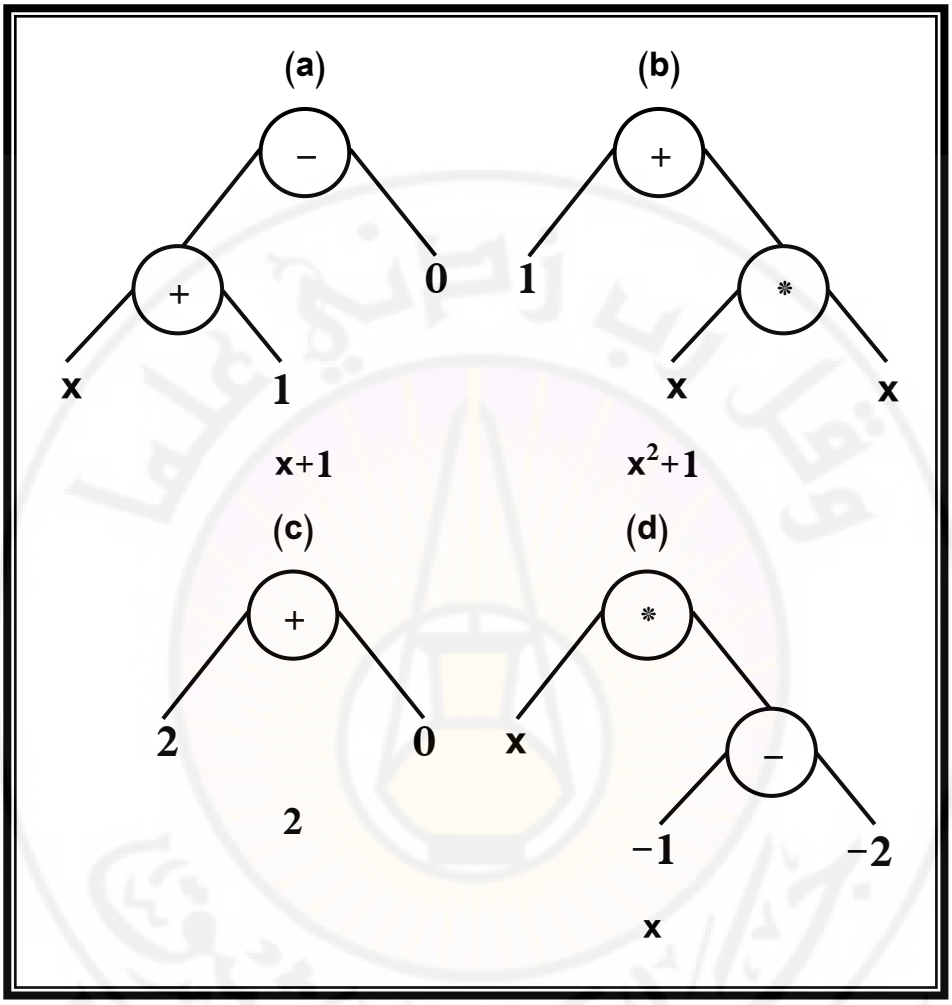
بالنسبة إلى تابع الملاءمة فيمكن بكل بساطة أخذ مجموع القيم المطلقة للفروق (The sum of absolute errors) عند جميع نقاط الجدول، بين القيم المبينة للتابع (y)، والقيم المحسوبة من التابع المستنتج من البرنامج. كلما كانت قيمة تابع الملاءمة أصغر في هذه المثال، كلما اقتربنا من الحل الأمثل. وعندما تصبح قيمة تابع الملاءمة صفراً، فهذا يعني الوصول إلى تابع يحقق تطابقاً كاملاً مع نقاط الجدول دون أي خطأ.

عملياً، عند حل المسائل المعقدة في البرمجة الجينية يقدر حجم الجيل بالآلاف، وحتى بالملايين. أما في هذا المثال البسيط، فإننا سنعتبر حجم الجيل أربعة فقط.

يفرض علينا الحجم الصغير للجيل والهدف التعليمي أن نتجاوز القيم العملية لكل من احتمال العبور والطفرة. ستنفذ عمليات العبور بنسبة قليلة لإنتاج حلين فقط. وسنقوم بعملية طفرة على إحدى الأشجار، متجاوزين احتمال الطفرة الذي يقدر بحوالي (1%)، والذي إذا ما نفذ على جيل بحجم صغير سيؤدي إلى فرصة منخفضة جداً في حدوث الطفرة.

يبقى لدينا تحديد شرط التوقف. بما أن تابع الملاءمة يتناهى للصفر عند الحل الأمثل فإننا سنكتفي بالوصول إلى دقة (0.1)، وهي مجموع القيم المطلقة للفروق بين القيم الحقيقية وقيم التابع المستنتج.

بعد تحديد المتطلبات السابقة والقيود، يجب توليد الجيل الابتدائي، والذي هو عبارة عن أربع أشجار عشوائية، وكل شجرة تمثل برنامجاً محاكياً لتابع صحيح بدرجة وأمثال مختلفة. يبين الشكل(6-47) الجيل الابتدائي مع التابع المطابق لكل شجرة.

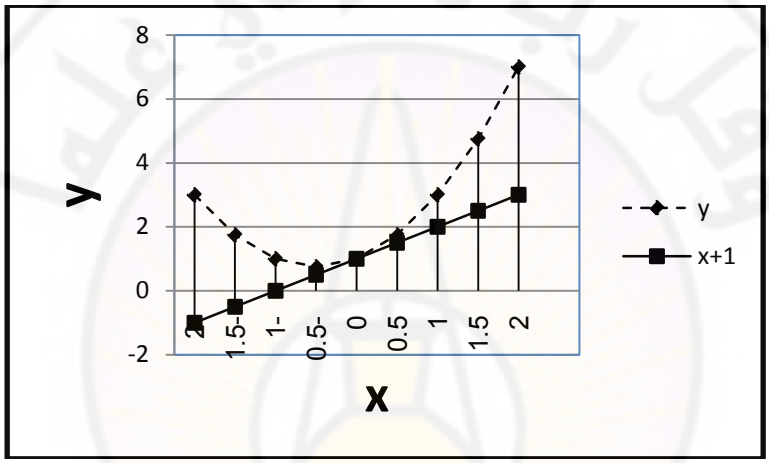


الشكل(6-47) أشجار الجيل الابتدائي الأربع

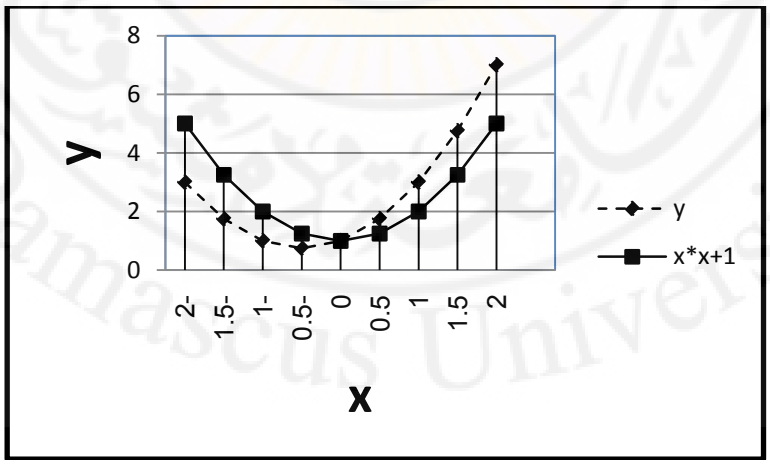
يحسب تابع الملاءمة لكل شجرة من الجيل الابتدائي، فمن الممكن أن تتحقق الصدفة المحضة، ويكون أحد أفراد الجيل الابتدائي هو الحل المطلوب المطابق لشرط التوقف. يبين الجدول الآتي قيم توابع الجيل الابتدائي عند قيم النقاط المدروسة.

x	−2	−1.5	−1	−0.5	0	0.5	1	1.5	2
y	3	1.75	1	0.75	1	1.75	3	4.75	7
x+1	−1	−0.5	0	0.5	1	1.5	2	2.5	3
|e|	4	2.25	1	0.25	0	0.25	1	2.25	4
sum of absolute errors= 15									
x^2+1	5	3.25	2	1.25	1	1.25	2	3.25	5
|e|	2	1.5	1	0.5	0	0.5	1	1.5	2
sum of absolute errors= 10									
2	2	2	2	2	2	2	2	2	2
|e|	1	0.25	1	1.25	1	0.25	1	2.75	5
sum of absolute errors= 13.5									
x	−2	−1.5	−1	−0.5	0	0.5	1	1.5	2
|e|	5	3.25	2	1.25	1	1.25	2	3.25	5
sum of absolute errors= 24									

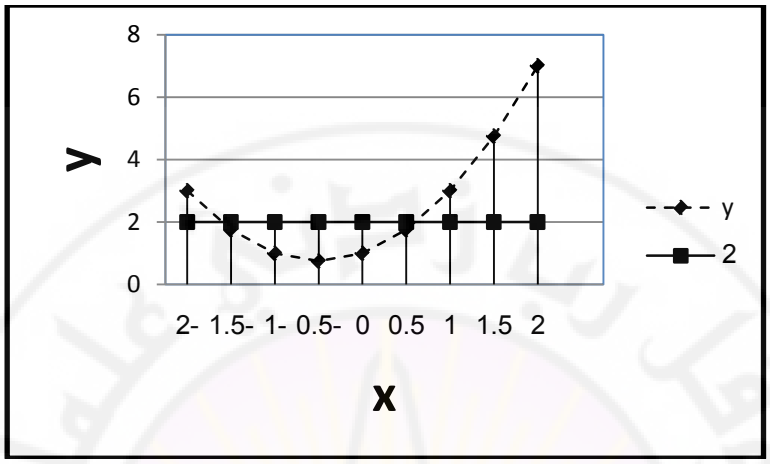
يتضح من الجدول أن التابع الرابع (x) هو الحل الأسوأ ضمن الجيل الابتدائي، لأن قيمة تابع الملاءمة له هي الأعلى. بينما التابع الثاني (x^2+1) هو الحل الأفضل ضمن الجيل الابتدائي، لأن قيمة تابع الملاءمة له هي الأقرب للصفر. تبين الأشكال (6-48) و(6-49) و(6-50) و(6-51) توابع الجيل الابتدائي مع النقاط المطلوب مطابقتها.



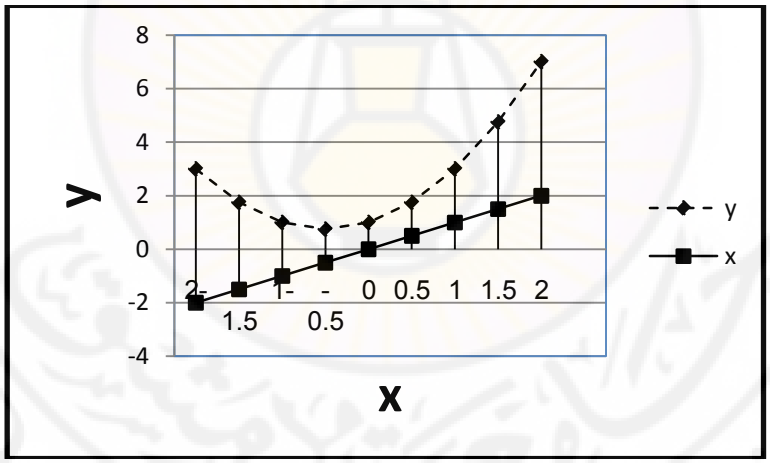
الشكل(6-48) التابع (x+1) من الجيل الابتدائي



الشكل(6-49) التابع (x^2+1) من الجيل الابتدائي



الشكل(6-50) التابع (2) من الجيل الابتدائي

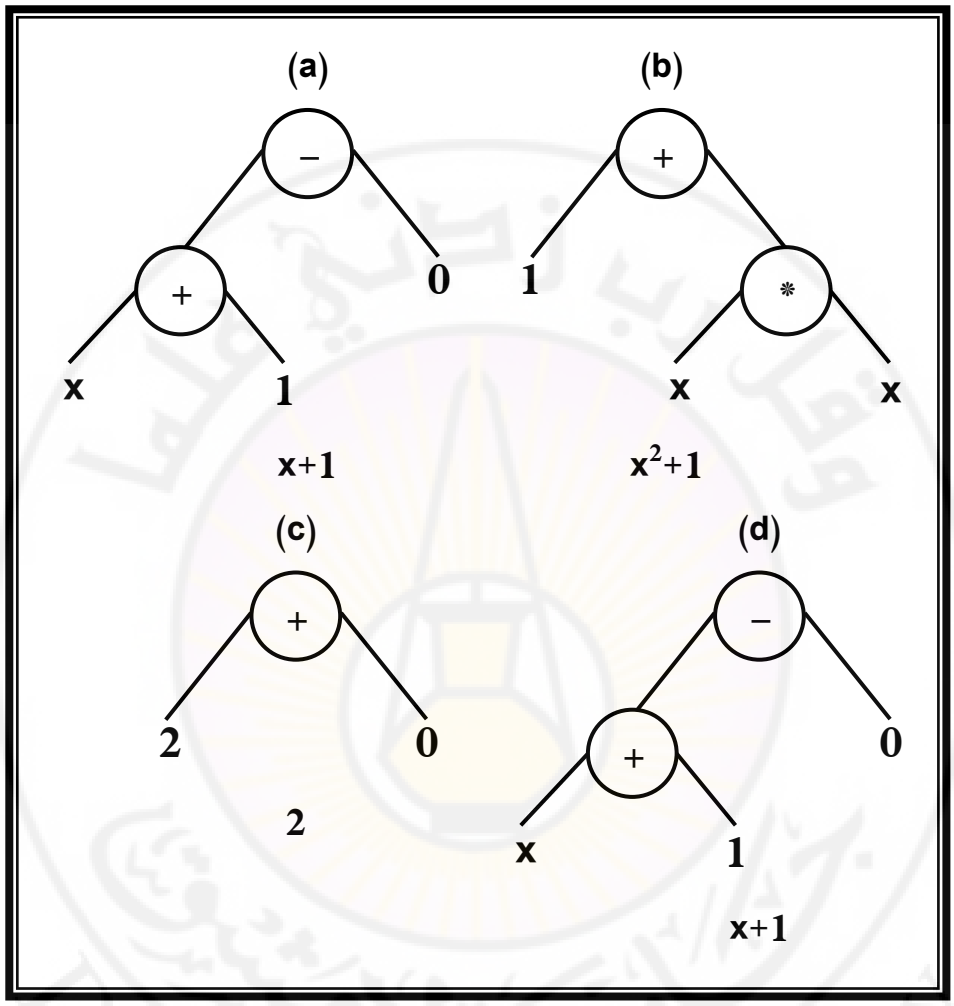


الشكل(6-51) التابع (x) من الجيل الابتدائي

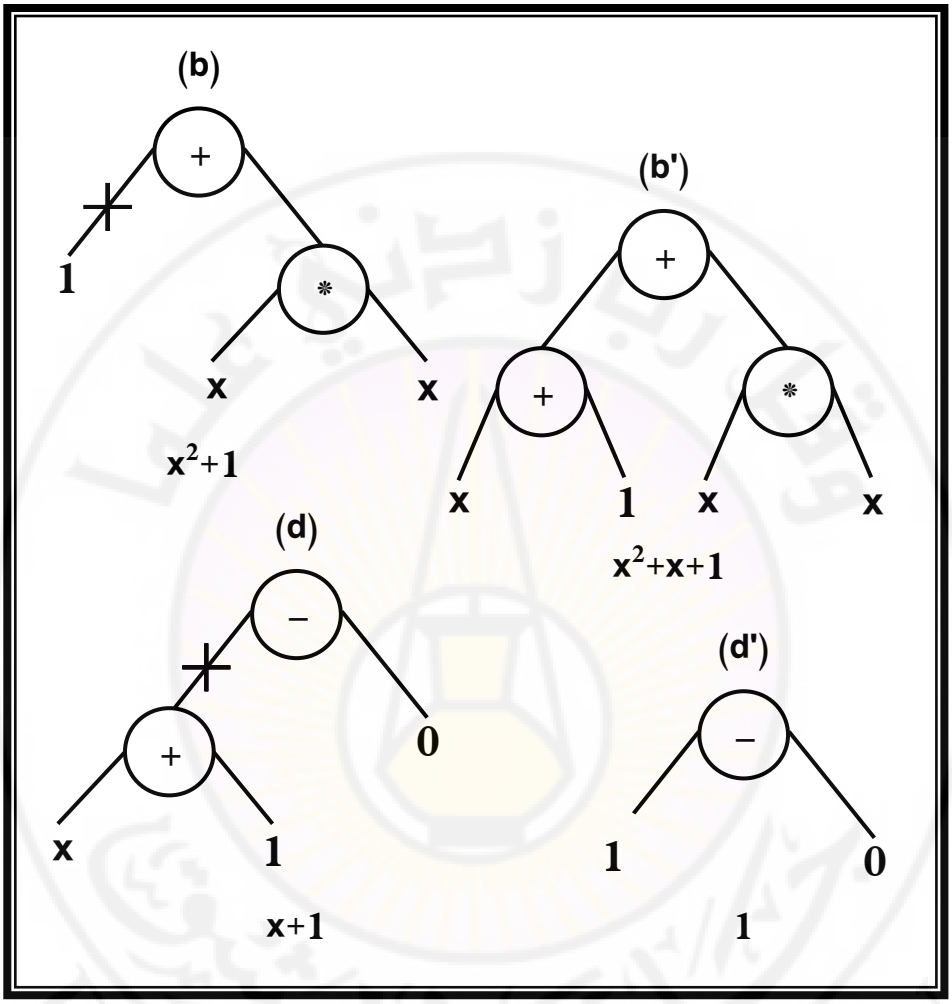
بما أن الجيل الابتدائي لا يحتوي على الحل المحقق للدقة المطلوبة (0.1)، نتابع العمليات المعروفة وهي الانتخاب والعبور والطفرة.

ليس من الضروري في عملية الانتخاب أن يتم انتخاب أفضل الحلول فقط، فمن الممكن أن لا ينتخب أفضل حل في الجيل. وأيضاً، من الممكن أن ينتخب أسوأ حل في الجيل. بما أن الهدف من ذا المثال البسيط هو التعرف على (GP)، فإننا سنفترض أنه تم انتخاب التابع الأول والثاني والثالث والأول أيضاً. يبين الشكل(6-52) التوابع المنتخبة بعد تنفيذ إحدى خوارزميات الانتخاب.

تنفذ عملية العبور بين بعض أفراد (حلول) الجيل وفق محدد احتمالي. سوف نعتبر تحقق المحدد الاحتمالي على الشجرة الثانية (b) والشجرة الرابعة (d). باختيار نقطة قطع عشوائية، فإننا نحصل على حلين، كما هو موضح في الشكل(6-53)، حيث نلاحظ تبادل فرعي الشجرتين في (b) و(d) للحصول على (b') و(d').

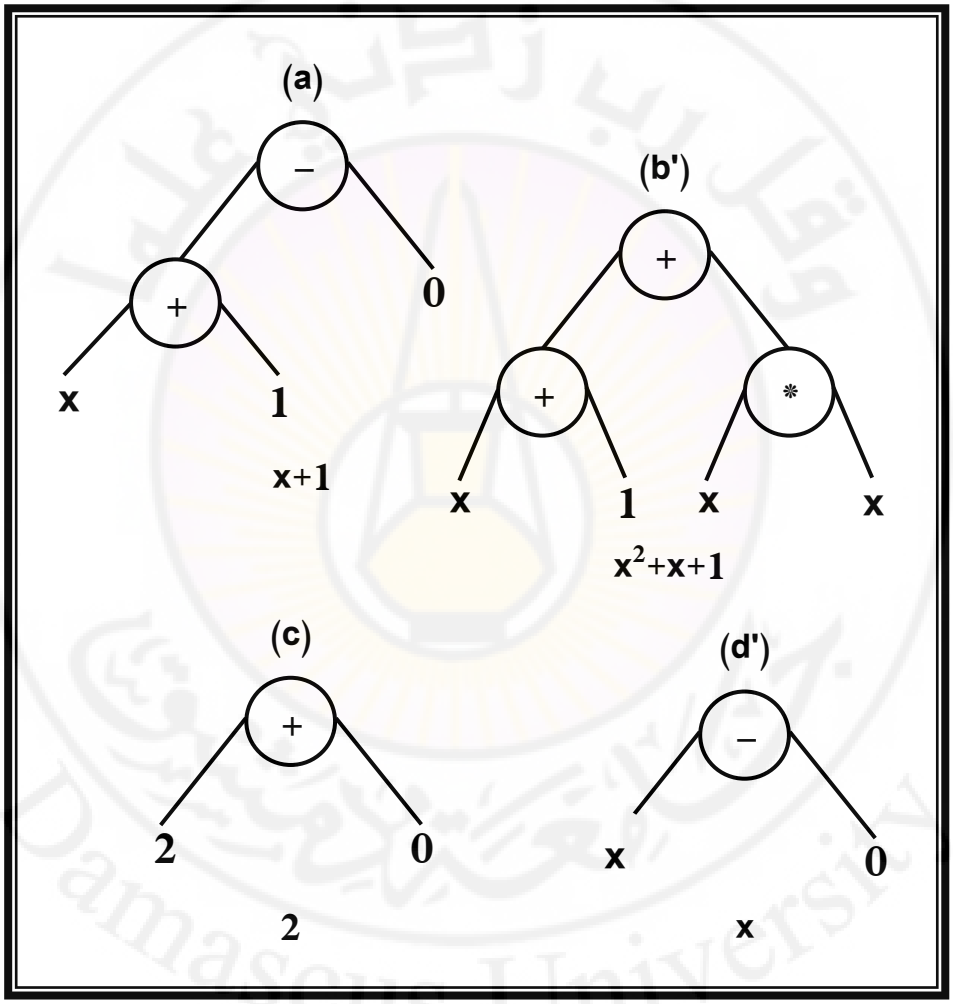


الشكل(6−52) التوابع المنتخبة



الشكل(6-53) عملية العبور

لتنفيذ الطفرة وهي المرحلة الأخيرة فإننا نعتبر أن احتمال الطفرة قد تحقق على الشجرة ('d) بتغيير العقدة الطرفية (1) إلى (x) وبالتالي يصبح جيل الأبناء الذي يجب اختبار شرط التوقف عنده هو المبين بالشكل(6-54).



الشكل(6-54) جيل الأبناء

يحسب تابع الملاءمة لكل شجرة من جيل الأبناء كما هو مبين في الجدول الآتي:

x	−2	−1.5	−1	−0.5	0	0.5	1	1.5	2
y	3	1.75	1	0.75	1	1.75	3	4.75	7
x+1	−1	−0.5	0	0.5	1	1.5	2	2.5	3
|e|	4	2.25	1	0.25	0	0.25	1	2.25	4
sum of absolute errors= 15									
x^2+x+1	3	1.75	1	0.75	1	1.75	3	4.75	7
|e|	0	0	0	0	0	0	0	0	0
sum of absolute errors= 0									
2	2	2	2	2	2	2	2	2	2
|e|	1	0.25	1	1.25	1	0.25	1	2.75	5
sum of absolute errors= 13.5									
x	−2	−1.5	−1	−0.5	0	0.5	1	1.5	2
|e|	5	3.25	2	1.25	1	1.25	2	3.25	5
sum of absolute errors= 24									

يتضح من الجدول أن التابع (x^2+x+1) هو الحل الأمثل، لأنه قيمة تابع الملاءمة له وصلت إلى قيمة الصفر. كما نلاحظ أن ا لتابع (x) نتج عن شجرة مختلفة في بنيتها عن الشجرة ذات التابع نفسه في الجيل الابتدائي.

وصلنا إلى الحل الأمثل خلال دورة واحدة فقط، وهذا ليس مطابقاً للحالة العملية التي تختلف عن المثال المقدم، والتي تحتاج إلى آلاف التكرارات.

6-15-8- تطبيقات في (GP):

تستخدم البرمجة الجينية في تطبيقات كثيرة ومتنوعة، فقد سجلت آلاف التطبيقات الهامة في عدة مجالات بحثية وعملية. سوف نستعرض بعض هذه التطبيقات بشكل مختصر.

1) مسائل الانحدار الخطي (Regression Problems):

من أقدم تطبيقات (GP) وأكثرها استخداماً، هي مسألة البحث عن تابع رياضي يمكن من خلاله التعبير عن العلاقة الناظمة لمجموعة من النقاط المأخوذة نتيجة تنفيذ إحدى التجارب العملية أو المخبرية، دون معرفة مسبقة بالعلاقة الرابطة بين دخل النظام وخرجه، بسبب إما صعوبة نمذجة النظام أو عدم توافر المعطيات الكافية. تنبع أهمية حل هكذا مسائل في مصادفتها في جميع مراكز الأبحاث على مستوى العالم، واستمرار وجودها ما دام هناك تجارب وأبحاث علمية.

تتواجد مجموعة من الطرائق الجيدة في حل هكذا نوع من المسائل، لكن عادة ما يتم تحديد الشكل العام للتابع المرغوب، كأن يطلب إيجاد أفضل تابع صحيح خطي، أو إيجاد أفضل تابع أسي، أو جيبي، أو أي شكل آخر معتمد في الأوساط العلمية.

تتميز (GP) عن غيرها من الطرائق المستخدمة في إيجاد التابع الممثل لمجموعة نقاط، بقدرتها على إيجاد التابع دون تحديد نمطه أو شكله العام. فمجرد توسيع مجموعة الطرفيات ومجموعة التوابع الممكن استخدامها في أداة (GP) لتشمل مثلاً (sin, cos, exp, log)، فهذا يعني البحث عن جميع الأنماط الجيبية والأسية والخطية والهجينة وغيرها.

2) المسائل التنافسية (Human Competitive Results):

يهدف الذكاء الصنعي إلى الحصول على نظام قادر على محاكاة أفعال البشر وردود أفعالهم وأسلوب تفكيرهم. قدم آلان تورنغ (Alan Turing) اختباراً يبين مقدار تحلي النظام الصنعي بالذكاء البشري، لكن هذا الاختبار غير عملي في بعض الحقول العلمية. قدم كوزا (Koza) اقتراحاً من مجموعة نقاط لاختبار نتائج البرامج المستنبطة بوساطة (GP) فيما إذا كانت على مستوى مقبول من الذكاء أم لا، وبذلك انتقل اختبار (Turing) إلى مفهوم آخر سهل التطبيق عملياً هو التنافس مع الإنسان (human competitiveness). تتمحور النقاط المقترحة من قبل (Koza)، حول أن تكون نتائج البرنامج الناتج عن أداة (GP)، مطابقة لنتائج أبحاث منشورة في مجلات علمية محكمة ودورية، أو أفضل منها. أو أن تكون مطابقة لبراءة اختراع مسجلة أو ابتكار موثق سابقاً أو نتائج يمكن أن ينتج عنها تسجيل حقوق ملكية. بالطبع، سجلت الكثير من البرامج التي اتصفت بما سبق، فمنها ما كان مطابقاً لأبحاث سابقة، ومنها ما سجل كبراءة اختراع جديدة، خاصة بعد رصد مكافأة مادية مغرية لكل من يقدم الجديد في هذا المضمار. تركزت معظم الأعمال في مجال الدارات الإلكترونية كتصميم دارات التضخيم والترشيح ورسم الدارات المطبوعة وتصميم الهوائيات وتحديد القيم المثلى لاستقرار أنظمة التحكم الصناعية.

3) معالجة الإشارة والصورة (Image and Signal Processing):

كان لعلم معالجة الإشارة أهمية بالغة في معظم المجالات الهندسية، ومع تطور الحواسيب فقد ازداد الاهتمام بهذا العلم، بالإضافة إلى بروز علم معالجة الصورة الذي استفاد كثيراً من التزايد المطرد في سرعة المعالجات، ليقدم لنا حلولاً

غاية في الدقة، سواء في المجالات العسكرية أم المدنية. كان لتطبيق (GP) فوائد عديدة في الحصول على حلول لمشاكل عدت معقدة لفترة طويلة في مجال معالجة الإشارة والصورة. كانت التطبيقات الصناعية والعسكرية من أوائل تطبيقات (GP) في معالجة الإشارة والصورة، فقد وضعت خوارزمية فحص وتحليل للإشارات المباشرة الناتجة عن المحركات الصناعية بغية دراسة إمكانية استخدامها في عمليات التحكم والمراقبة لأداء المحرك. كما تم الاستفادة من الصور المأخوذة بالأشعة تحت الحمراء لكشف مواقع الآليات العسكرية الثقيلة. لم تقتصر التطبيقات العسكرية على البر، بل تعدتها إلى التطبيقات البحرية، بكشف السفن وغيرها باستخدام الأمواج فوق الصوتية. كما كان للتطبيقات الجيولوجية أهميتها في تحليل بعض أنواع الصور للتعرف على أماكن تواجد الخامات الثمينة. قام بعض الباحثين بدمج إمكانيات الشبكات العصبونية مع إمكانيات (GP) لدراسة خصائص المرشحات التمثيلية في الدوائر الإلكترونية. أيضاً، استخدمت (GP) في التعرف على الوجوه، وفي تركيب الصور ثلاثية الأبعاد انطلاقاً من صورتين ثنائيتي البعد من زاويتين مختلفين للمجسم، وفي أنظمة التعرف الآلي على المحارف المطبوعة أو المكتوبة بخط اليد.

4) عمليات التحكم الصناعية (Industrial Process Control):

لا مجال لحصر استخدامات (GP) في المجال الصناعي، فبعضهم استخدمها في دراسة تخفيض الكلف المادية خلال عمليات التشغيل، وبعضهم الآخر استخدمها في محاكاة المفاعلات النووية للوصول إلى أعلى درجات الأمان. كما كان لعلم الكيمياء وخاصة ما يتعلق بخلائط البلاستيك، وافر النصيب من

الدراسات المتعلقة بعمليات تصنيعه وتشكيله، والحصول على خلائط بمواصفات محددة باستخدام (GP).

بالإضافة إلى المجالات سابقة الذكر، فإن للبرمجة الجينية استخدامات أخرى هامة، كاستخدامها في الدراسات السكانية، والمجالات الطبية، وعلم الأحياء، ونظرية الألعاب والتسلية، وعلوم الحاسوب، والعلوم الإنسانية والفنية، والتجارة والاقتصاد، والتسويق والمحاسبة، والتنقيب عن المعطيات وفهرستها.

الفصل السابع

7- لغة البرمجة البرولوغ (PROLOG):

7-1- مقدمة:

يسمو الإنسان بعقله الذي يتمتع بقدرة كبيرة على التفكير وحل المسائل اعتماداً على ذاكرة قوية، وكم هائل من المعلومات والعمليات المعقدة. حاول العلماء تقليد ذكاء البشر بوساطة نمذجة عملية الإدراك والتفكير والتخزين على الحاسوب للوصول إلى الذكاء الصنعي (Artificial Intelligence) بتقاناته المتنوعة. للذكاء البشري أفعال عديدة كالإدراك والوعي والمعرفة والفهم والاستيعاب والاستنتاج والتنبؤ، حيث تدل هذه الأفعال على قدرة تفكير بآلية معقدة للغاية.

تعد لغة البرمجة البرولوغ من أكثر لغات البرمجة المخصصة لأبحاث الذكاء الصنعي انتشاراً، وقد اشتقت من الجملة (Programming in Logic).

انطلقت البرولوغ من جامعة مرسيليا (Marseille) في فرنسا في سبعينيات القرن الماضي بهدف معالجة اللغات الطبيعية لذلك فإن تسميتها الفرنسية هي (Programmation et Logique).

لم تكن البرولوغ هي لغة الذكاء الصنعي الأولى أو الوحيدة فقد سبقتها لغة (IPL) التي كانت أولى اللغات المصممة لتطبيقات الذكاء الصنعي. كما تتواجد مجموعة من اللغات مثل (LISP, STRIPS).

يمكن القول: إن لغات البرمجة تقسم إلى مجموعتين مختلفتين؛ هما:

1. لغات برمجة خرجها برمجي:

يكون ناتجها النهائي عبارة عن ترميز ثنائي مخزن ضمن الحاسوب، وهو يعبر عن تعليمات برمجية تهدف لتنفيذ أوامر محددة. يمكن لهذا الترميز أن ينتج نظام تشغيل أو برنامجاً تطبيقياً أو لغة برمجية أو برنامجاً لإدارة قواعد المعطيات.

2. لغات برمجة خرجها مادي:

يكون ناتجها النهائي عبارة عن قطعة مادية أو دارة متكاملة (FPGA)، والتي تعني مصفوفة البوابات المنطقية القابلة للبرمجة (Field Programmable Gate Array). تحتوي هذه القطعة المادية على عدد كبير من العناصر المنطقية، والتي يتم توصيلها فيزيائياً حسب البرنامج المكتوب لها، لتعمل وفق منهجية محددة ولمهمة واحدة عموماً. يوجد من هذا النوع عدة لغات مثل (VHDL) وهي مشتقة من الجملة (Very High Speed Integrated Circuits Hardware Description Language)، ولغة (Verilog)، ولغة (ABEL) وهي مشتقة من الجملة (Advanced Boolean Equation Language).

بالنسبة إلى المجموعة الأولى فهي الأكثر انتشاراً واستخداماً، حيث تقسم بدورها إلى مجموعتين مختلفتين؛ هما:

1. لغات برمجة عالية المستوى:

هذا النوع هو الأكثر انتشاراً والأقرب إلى لغة الإنسان الطبيعية في إصدار الأوامر.

2. لغات برمجة متدنية المستوى:

تعد أصعب أنواع اللغات، وأكثرها قدرة على التحكم بالكيان الصلب في الحاسوب، ولكنها تتطلب عدداً كبيراً من الأسطر البرمجية لتنفيذ مهام بسيطة. انخفضت الحاجة في السنوات الأخيرة لهذا النوع بسبب التطور في لغات البرمجة عالية المستوى، والحاجة إلى برامج ذات إمكانيات عالية وخيارات متنوعة. تختلف هذه اللغات حسب المعالج الذي تتم عليه عملية البرمجة وتدعى بلغة التجميع

(Assembly)، والتي يتم تحويلها إلى لغة رقمية هي لغة الآلة (Machine Code) ذات الاتصال المباشر بالمعالج.

أيضاً، بالنسبة إلى لغات البرمجة عالية المستوى فهي تقسم بدورها إلى مجموعتين مختلفتين: هما:

1. لغات تقليدية:

تكتب معظم البرمجيات الحالية باستخدام لغات تقليدية عالية المستوى، وهي ذات تنوع وعدد كبيرين، وتستخدم في كثير من التطبيقات البرمجية الخدمية والصناعية والبحثية . بعض هذه اللغات موجهة لتطبيقات رياضية، ومنها لغات موجهة لتطبيقات الشبكات، ومنها لغات مرئية، وأخرى تعتمد على أسطر الأوامر، وغيرها الكثير. كمثال على هذا النوع من اللغات نذكر، (C) و(C++) و(Delphi) و(Python) و(Java) و(R).

2. لغات الذكاء الصنعي:

تتجه معظم التطبيقات الحديثة نحو الاستفادة القصوى من علم الذكاء الصنعي، خاصة عند التفكير بالجيل السادس للحواسيب وإنترنت الأشياء وغيرها. رغم صلاحية اللغات التقليدية لكتابة برامج ذكية، إلا أنه من الأفضل استخدام لغات برمجية موجهة لأغراض الذكاء الصنعي عند كتابة برامج تتمتع بما يشبه ويقترب من الذكاء البشري. أوجدت أولى لغات الذكاء الصنعي لمعالجة اللغة الطبيعية عند الإنسان وهي لغة (IPL) ثم انتشر عدد محدود من لغات الذكاء الصنعي مثل (LISP, STRIPS). أما اللغة التي اخترت في هذا الكتاب فهي لغة (Prolog)، واسعة الانتشار والاستخدام.

يكمن الفرق الجوهري بين لغات البرمجة التقليدية، ولغات الذكاء الصنعي فيما يأتي:

إن اللغات الأكثر انتشاراً هي اللغات التقليدية التي توصف بأنها لغات إجرائية (Procedural)، أي تقوم بتنفيذ خوارزمية واضحة وبتفرعات محددة وفق شروط معروفة. وبالتالي، تكتب البرامج التي تقوم بالأرشفة أو استرجاع البيانات أو المستخدمة لتسريع الوصول لحل المسائل المطروحة معروفة الحل باستخدام اللغات التقليدية. فمثلاً، يمكن كتابة برنامج بإخراج أنيق بأي لغة إجرائية يستخدم لحل المعادلات من الدرجة الثانية، وذلك بإدخال أمثال المتحول من الدرجة الأولى والثانية والثابت، ليقوم البرنامج بعدها بإتباع خوارزمية معروفة تكون نتيجتها النهائية حلول المعادلة الصحيحة.

أما لغات الذكاء الصنعي فتوصف بأنها لغات وصفية تصريحية (Declarative)، فهنا نعرف المشكلة ، لكننا لا نعرف طريقة حلها، وبدلاً من تحديد طريقة الحل وكيفية الوصول إليه فإننا نوصف المشكلة باستخدام مجموعة من الحقائق والعلاقات المنطقية، ونترك مفسر البرولوغ (interpreter) ليقوم بدوره بإيجاد حلول المسألة المطروحة بالطريقة المناسبة ، ودون تحديد ذلك من قبل المبرمج. وبالتالي فالمبرمج يحدد للحاسب كيف عليه أن يحل المسألة في اللغات الإجرائية، لأنه يعرف المسألة وكيفية حلها مسبقاً . أما في اللغات الوصفية فما عليه سوى أن يطرح المسألة التي لا يعرف حلها مسبقاً، ويطلب من الحاسب إيجاد الحل وبصرف النظر عن الطريقة.

عموماً، هناك تكامل في نوعي لغات البرمجة ، بحيث لا يمكن الاستغناء عن أحدهما، فلكل نوع مجال استخدامه وأفضليته على النوع الثاني. فعندما نفكر بكتابة برنامج لمعالجة قاعدة معطيات ضخمة، أو تصميم مواقع على صفحة الإنترنت، أو كتابة برنامج لحل بعض المسائل الرياضية ذات المتحولات العقدية، أو حتى عندما نريد التعامل مع الوسط الخارجي والتحكم بخطوط إنتاج صناعية،

عندها لن نكون موفقين باختيار إحدى اللغات الوصفية، ولا بد للحصول على منتج برمجي جيد من التعامل مع اللغات الإجرائية. أما عندما نود كتابة برنامج للبحث عن حل لبعض الألغاز الذهنية، أو البحث عن علاقات غير معروفة بين عناصر المسألة، أو حتى عندما نرغب ببرمجة إنسان آلي (Robot) يمكنه التحاور باللغة الطبيعية مع البشر مع مراعاة القواعد النحوية، عندها فمن المستحسن اختيار إحدى اللغات الوصفية، فهي الأفضل لهذا النوع من المسائل.

7-2- تمثيل الحقائق (Representing Facts):

لكي يعمل الحاسوب بطريقة ذكية، يجب إعطاؤه المعرفة (Knowledge) كي تتكون لديه صورة واضحة عما نعرف، وهذا الأمر ليس بالسهل أبداً. يوجد صنفان أساسيان للمعارف هما الحقائق (Facts)، والاستدلالات (Inferences). تشير الحقائق إلى أشياء صحيحة ومعلومات حقيقية لا تناقش كالمسلمات والفرضيات. أما الاستدلالات والاستنتاجات فهي طرائق الحصول على معرفة جديدة من الحقائق.

تتضمن لغة البرولوغ مكاناً لكتابة البرنامج يسمى (Program)، ومكاناً للاستعلام يسمى (Query). وبالتالي فإن المبرمج يقوم بكتابة النص البرمجي ضمن المكان المخصص له، ليخزنه ضمن ملف نصي بلاحقة (pl.*). بعد ذلك يقوم بالاستعلام من المكان المخصص للاستعلام، بكتابة الأسئلة المناسبة وفق القواعد الناظمة لها، ليقوم البرولوغ بإعطاء الإجابة المناسبة مباشرة وبالشكل المرغوب، مع إمكانية الحصول على جميع الحلول الممكنة للسؤال المطروح الذي ربما يكون حلاً لمسألة معقدة.

يتضمن النص البرمجي قاعدة المعطيات (Database) الواصفة للمسألة بشكل محدد ودقيق، وهي تتضمن المعلومات المعروفة عن المسألة والتي تصفها بشكل معطيات صحيحة هي الحقائق (Facts). كما يمكن أن تتضمن قاعدة المعطيات مجموعة من العلاقات العامة التي تربط عبارات الحقائق فيما بينها، وتجعل البرنامج أكثر نفعاً وأفضل أداءً، بتقليل عدد كبير من الحقائق والاستعاضة عنها بالعلاقات (Rules). يجب الانتباه إلى أن الحقائق تكون صحيحة دائماً، أما القواعد فتكون صحيحة بشكل مشروط.

تعتمد البرولوغ على التعابير الإعلانية (Predicate Expressions)، وهي عبارة عن اسم يدعى معلن أو مسند (Predicate)، متبوع بعدد من المعاملات أو الوسطاء (Arguments)، المفصولة عن بعضها بفاصلة وموضوعة ضمن قوسين.

فمثلاً عند كتابة ((city(damascus)، فهو تعبير إعلاني يتضمن اسم معلن هو (city)، ووسيط هو (damascus)، وتوصف العبارة بأنها حقيقة (Fact)، وتعني العبارة ببساطة أن دمشق مدينة. أما عند كتابة ((x(7))، فهي حقيقة تعني أن قيمة (x) تساوي (7).

يجب مراعاة استخدام الأحرف الصغيرة (Small Letters) في تسمية المعلنات، واختيار الأسماء بعناية بحيث تشير إلى المعنى الحقيقي. كما يمنع استخدام أسماء محجوزة في اللغة لتعريف معلن جديد كاستخدام (append) مثلاً، المحجوزة للتعامل مع القوائم. عموماً، فإنه يفضل إتباع الاسم برقم لضمان عدم تعريفه مسبقاً في اللغة كتعريف (append1). كما يفضل إعطاء المعلنات أسماء أكثر شمولية من أسماء الوسطاء، فاسم المركبة أكثر شمولية من السيارة.

ليس من الضروري دائماً استخدام الوسطاء مع المعلنات، فمن الممكن تعريف معلن دون وسطاء، ولكنها ستكون حقيقة دون استخدام عملي في البرنامج، وكمثال عليها كتابة أي كلمة منفردة:

fine.

فهي حقيقة عامة تعني أن كل شيء رائع. من الهام إنهاء أي سطر في البرولوغ بنقطة، سواء كان حقيقة أم قاعدة، أم استفساراً .

يمكن أن يكون عدد الوسطاء واحداً أو اثنين أو أكثر، مثل:

father_name(samer, asaad).

هي حقيقة تعني أن أسعد أبو سامر، وقد تم استخدام الخط السفلي للفصل بين كلمتين في تسمية المعلن. ولكن من يحدد أن الوسيط الثاني هو الأب، والوسيط الأول هو الابن؟. إن الذي يحدد ذلك هو المبرمج، والذي يقوم بتوضيح ذلك للمستخدم في دليل استخدام البرنامج. يجب أن يلتزم المبرمج أيضاً بهذه القاعدة في جميع أجزاء البرنامج، لأنه يجوز تكرار اسم المعلن كما سنرى لاحقاً.

في الحقيقة الآتية:

area(syria, 185180).

نبين إمكانية استخدام الأرقام في تعريف الحقائق، وهي حقيقة تعني أن مساحة سورية تساوي ($185180\ Km^2$).

يسمح بتكرار الحقائق، أو استخدام اسم المعلن أكثر من مرة، سواء كاسم حقيقة أو قاعدة.

day_of_week(friday).

day_of_week(sunday).

كما يمكن استخدام الوسيط في أكثر من معلن مختلف:

first_day_week(sunday).

كما يمكن استخدام الوسيط كاسم معلن آخر:

syria_uni(damas_uni).

damas_uni(damascus).

بالطبع لغة البرولوغ لا تعرف أنه في سورية يوجد جامعة اسمها جامعة دمشق، وأن جامعة دمشق موجودة في مدينة دمشق، فجميع أسماء المعلنات هذه، مع الوسطاء، مجهولة بالنسبة للغة، لكن وجود الحقائق كتعاريف للكلمات المستخدمة تؤدي إلى الإحاطة بالمعنى. فمجموعة الحقائق كاملة في البرنامج هي التي ستؤدي إلى توليد استدلالات جديدة تؤدي إلى انبثاق الذكاء.

إذن، البرولوغ يربط الوسطاء مع المعلنات بعلاقات دون معرفة معنى التعبير الإعلاني.

7-3- أنماط المعلنات (Predicates Types):

تقسم معلنات البرولوغ إلى ستة أنماط هي:

1. معلن النوع (Type Predicate):

يشير إلى شيء ما، ويستخدم لتحديد صنف أو نوع الشيء، وعدد وسطائه (1). في المثال الآتي نحدد أن الأمل هي سفينة رمادية اللون وكبيرة الحجم. أو يمكن القول: إن الأمل هي إحدى أنواع السفن، وإحدى الأشياء الرمادية، وإحدى الأشياء الكبيرة.

ship(al_amal).

gray(al_amal).

big(al_amal).

2. معلن الخاصة (Property Predicate):

يشير إلى شيء ما وخاصيته، ويستخدم لتحديد خواص الشيء، وعدد وسطائه (2). في المثال السابق استخدم المعلن (gray) و(big)، لكنهما لا يتمتعان بخاصية الشمولية في تسمية المعلنات، لذلك يمكن كتابة ما يأتي:

```
color(al_amal, gray).   % Property Predicate
size(al_amal, big).   % Property Predicate
```

نلاحظ إمكانية وضع اسم المعلن بين الوسيطين مع فعل الكون (is) للحصول على جملة إنكليزية ذات معنى.

تستخدم إشارة النسبة المئوية (Percentage Sign %) لكتابة التعليقات على السطر نفسه، بوضعها في بداية كل سطر تعليق. عندما يتطلب التعليق أكثر من سطر، فيمكن كتابته ضمن الإشارتين (/* ... */).

```
size(al_amal, big).   /* This is a multi-line comment,
which must be closed with a */
```

3. معلن العلاقة (Relationship Predicate):

تشير الوسطاء إلى شيئين بينهما علاقة، وعدد وسطائه (2). نلاحظ هنا إمكانية وضع اسم المعلن بين الوسيطين للحصول على جملة إنكليزية ذات معنى.

```
part_of(fmee, damas_uni).
part_of(damas_uni, mohe).
part_of(mohe, syrian_gov).
owns(husam, house).
owns(husam, car).
a_kind_of(ship, vehicle).
```

a_kind_of(car, vehicle).

كثيراً ما يستخدم المعلن (a_kind_of) في معلن العلاقة.

4. معلن قاعدة البيانات (Database Predicate):

يشير إلى شيء وخواص الشيء، وعدد وسطائه (1) أو أكثر. يشبه سجل المعطيات ذي الحقول المتعددة.

ship(al_amal, gray, big, p125).

يشير الوسيط الأول إلى اسم المعلن وهو السفينة، والوسيط الثاني إلى لونها، والوسيط الثالث إلى حجمها، والوسيط الرابع إلى رقمها التسلسلي، ولا يجوز تغيير تسلسل الوسطاء في البرنامج الواحد.

5. معلن الوظيفة (Function Predicate):

يكون الوسيط الأخير هو ناتج تطبيق عملية حسابية، أو منطقية، أو أي عملية موصوفة بتابع ما، بين بقية الوسطاء، وبالتالي فإن عدد وسطائه (2) أو أكثر. ليس من الضرورة أن يتم وصف تابع عددي، فمن الممكن أن يكون الوسيط الثالث هو ناتج دمج الوسيطين الآخرين كمعلن (append) لدمج القوائم.

sum(7, 3, 4, 14).

sqr(64, 8).

6. معلن الاحتمال (Probability Predicate):

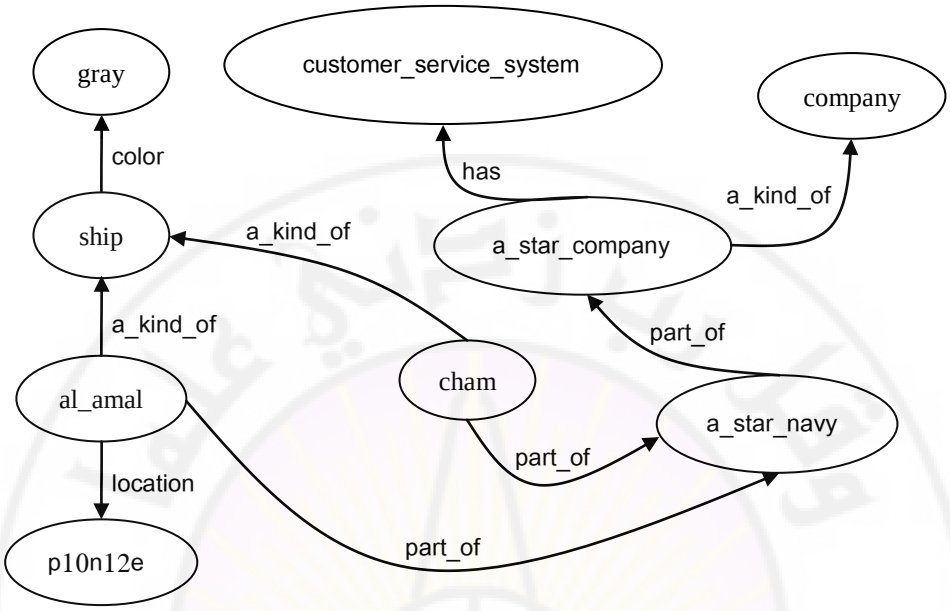
يكون الوسيط الأخير هو احتمال حقيقة ما، وبالتالي فإن عدد وسطائه (2) أو أكثر.

ship(al_amal, gray, 0.8).

أي أن احتمال أن يكون لون سفينة الأمل رمادياً هو (0.8).

تستخدم شبكات الدلالة (Semantic Networks) لتبسيط عملية فهم التعابير الإعلانية ذات الوسيطين. يرمز للوسيط بدائرة أو شكل بيضوي، وللمعلن بسهم ينطلق من دائرة الوسيط الأول ويستقر في دائرة الوسيط الثاني، مع كتابة اسم المعلن على السهم. يبين الشكل(7-1) شبكة الدلالة للبرنامج الآتي:

```
a_kind_of(al_amal, ship).
a_kind_of(cham, ship).
part_of(al_amal, a_star_navy).
part_of(cham, a_star_navy).
part_of(a_star_navy, a_star_company).
a_kind_of(a_star_company, company).
color(ship, gray).
location(al_amal, p10n12e).
has(a_star_company, customer_service_system).
```



الشكل(7-1) شبكة الدلالة للبرنامج

7-4- بنية المعطيات (Data Structure):

تدعى بنية المعطيات الأساسية في البرولوغ العبارة أو المعرف (Term) وهي أربعة أنواع (Atoms, Numbers, Variables, and Compound terms)، وعند استخدام (Atoms and Numbers) سوية فإنها تسمى (Atomic Terms). وفيما يأتي الأنواع الأربعة للمعطيات:

1. الذرة (Atom):

عبارة عن سلسلة من الأحرف الإنكليزية الصغيرة والكبيرة والأرقام والخط السفلي (_ Underscore)، على أن تبدأ دائماً بحرف صغير، ومن أمثلتها:

elephant, b, abcXYZ, x_123, another_pen_for_me

يمكن استخدام أي سلسلة أخرى من المحارف ، ولو بدأت بحرف كبير ، على أن نضعها ضمن علامة اقتباس مفردة (Single Quotes)، مثل:

'This is also a Prolog atom.'

يمكن استخدام عدد من المحارف الخاصة مثل (+, −, *, =, <, >, :, &) لتعريف (Atoms) ولكن بشرط أن تكون لوحدها ، أي دون أي حرف آخر ، فمثلاً الأسماء الآتية مقبولة:

+, ::, <------>, ***

bigger(tiger, **>:).

مع الانتباه إلى أن بعض العبارات غير مسموحة عموماً، كبدء العبارة بحرف كبير، أو برقم، أو استخدام بعض المحارف الخاصة كالمسافة والناقص:

Hello , 4hello _Hello , two words , two-words

2. العدد (Number):

عبارة عن سلسلة من الأرقام، سواء كتبت لوحدها، أم سبقت سبقت بإشارة جبرية سالبة أو موجبة، أي ناقص (Minus) أو زائد (Plus). علماً أن البرولوغ يحافظ على إشارة الناقص إن وجدت، ويحذف إشارة الزائد في الإظهار، ومن أمثلتها:

bigger(thing, 567).

length(line12, 100).

3. المتحول (Variable):

عبارة عن سلسلة من الأحرف والأرقام والخط السفلي (_ Underscore)، على أن تبدأ دائماً بحرف كبير، أو بالخط السفلي، ومن أمثلتها:

X, Elephant, _4711, X_1_2, MyVariable, _

إن المتحول الأخير، أي الخط السفلي الوحيد، يمثل حالة خاصة، يدعى المتحول المجهول (Anonymous Variable)، ويستخدم عندما تكون قيمة المتحول غير مهمة (Do not Care).

4. العبارة المركبة (Compound Term):

تتكون من مجموعة مختلطة مما سبق (Functor)، تفصلها عن بعضها فاصلة عادية، مع عدم ترك أي فراغ، حتى بعد القوس مباشرة، لتكون عبارة في البرولوغ تدعى (Predicates)، وعند عدم احتوائها على أي متحول تسمى (A ground term)، وكمثال عليها:

is_bigger(horse, X), f(g(X, _), 7), 'My Func'(sheep)

7-5- بنية البرنامج (Program Structure):

يتضمن برنامج البرولوغ مجموعة حقائق (Facts) وقواعد (Rules) تسمى عبارات (Clauses)، ولنفصل قليلاً في صياغة هذه العبارات، وكيفية الاستعلام عنها.

1. الحقائق (Facts):

الحقيقة هي معلن (Predicate) متبوعاً بنقطة. وبما أنه معلن فيمكن أن يكون بأبسط شكل كلمة واحدة، أو عبارة تترجم لصيغة نعتمدها، فمثلاً:

```
fine.
life_is_beautiful.
father(omar).
bigger(whale, _).
```

كلها حقائق صحيحة من حيث الصياغة، وسوف يعتمدها البرولوغ كذلك. مع التوضيح إلى أن العبارة الأخيرة تعني أن الحوت الضخم أكبر من أي حيوان في قاعدة المعطيات. إن المعلن هو الاسم تحديداً (fine, life_is_beautiful, father, bigger).

من الهام الانتباه إلى أن المبرمج هو من يحدد كيفية ترجمة ما يكتب للبرولوغ، فإذا كتب:

sleep(tamim, bed, hours_8).

هذه الحقيقة يمكن ترجمتها بإحدى الشكلين الآتيين:

- ينام تميم في سريره ثماني ساعات.
- ينام تميم في سريره حتى الساعة الثامنة.

تكون الترجمة الصحيحة هي بسؤال المبرمج عن المعلن (sleep). واعتماد القاعدة التي استخدمها. ويجب الاستمرار بالترجمة نفسها في بقية المعلنات (sleep) في البرنامج الواحد. فالبرولوغ لا يعرف سوى العلاقات، أما كلمة نوم أو سرير أو تميم فهي عديمة المعنى بالنسبة له، وينحصر دوره في إنشائه لقضية اسمها (sleep) تربط بين (tamim)، وبين (bed)، وبين (hours_8)، وبصرف النظر عن ماهية العلاقة.

2. القواعد (Rules):

يتحقق الذكاء في برنامج البرولوغ من خلال الاستدلالات (Inferences) والخلاصات المستمدة من الحقائق، لا من الحقائق ذاتها. تهدف القواعد إلى مساءلة معلنات جديدة دون الحاجة إلى تعيين حقائق جديدة، وذلك بتعريف معلنات جديدة من خلال المعلنات القديمة.

تقسم القاعدة إلى طرفين، يدعى الطرف اليساري رأس القاعدة (Head) وله اسم معلن مع وجود عدة وسطاء، أو بدون وسطاء. ويدعى الطرف اليميني جسم القاعدة (Body) وله اسم واحد معلن أو أكثر، على أن تفصل بين معلنات الجسم فاصلة عادية (, Comma) أو فاصلة منقوطة (; Semicolon)، حسب ما هو مطلوب من التعبير المركب. بينما تفصل بين الرأس والجسم إشارة (:-) وهي تعني "إذا" أو "فيما إذا". تنتهي عبارة القاعدة في نهاية الجسم بنقطة. وكمثال عليها:

```
is_smaller(X, Y) :-
        is_bigger(Y, X).
```

حتى تصبح عبارة القاعدة هذه صحيحة، يجب أن يكون قد ورد المعلن (is_bigger) سابقاً، وهذا يذكرنا بتعريف المتحول قبل استخدامه في اللغات التقليدية. تترجم القاعدة بشكل لغوي بأنه يكون المتحول (X) أصغر من المتحول (Y)، فيما إذا كان، أو بشرط كون المتحول (Y) أكبر من المتحول (X)، في المثال الآتي:

```
mother(rana, omar).
parent(P1, P2) :-
        mother(P1, P2).
```

تم تعريف المعلن (mother) كحقيقة مسبقاً، فأصبح بالإمكان استخدامه ضمن جسم القاعدة.

بالانتقال إلى مثال آخر، دون التدقيق على الملاحظة الأخيرة:

```
aunt(Aunt, Child) :-
        sister(Aunt, Parent),
        parent(Parent, Child).
```

تكون عبارة القاعدة هنا، أكثر من معلن في الجسم، تفصلهما فاصلة عادية (,)، وبالتالي فالعلاقة (و). استخدمت كلمات ذات معنى كمتحولات، وهذا فقط لسهولة فهم البرنامج، فترجمة القاعدة نثرياً بالشكل:

يكون المتحول (Aunt) عمة للمتحول (Child) فيما إذا تحقق شرطان هما:

- المتحول (Aunt) هو أخت لمتحول آخر اسمه (Parent).
- المتحول (Parent) هو والد للمتحول (Child).

ويلاحظ أنه باستخدام معلن اسمه (parent) ومراعاة ذلك في بقية البرنامج، نكون قد عرفنا العمة والخالة سوية، حيث يعني الوالد، أحد الوالدين، أي الأب أو الأم.

يتحقق الطرف الأيسر من القاعدة عندما يتحقق الطرف الأيمن، وبالتالي عند عدم وجود طرف أيمن فهذا يعني أن القاعدة محققة دوماً. بما أن الشيء المحقق دوماً هو الحقيقة، فنستنتج أن القاعدة دون طرف أيمن هي حقيقة، والحقيقة حالة خاصة من القاعدة، ولذلك يكتبان سوية في البرنامج.

حتى عند وجود متحول في رأس القاعدة دون وجود لجسم القاعدة فسيكون المتحول محقق دوماً. عند كتابة القاعدة الآتية:

```
part_of(X, universe).
```

فهي تعني أن كل (X) هو جزء من الكون. وبالتالي فالحقائق ذات المتحولات ودون وجود جسم للقاعدة هي كميات شاملة.

تتميز لغات البرمجة عالية المستوى بقربها من اللغة الحية وإمكانية تحويل الجمل المحكية إلى ترميز برمجي بسهولة. إن أكثر قواعد اللغة الإنكليزية تداولاً هي (If .... then)، فعلى سبيل المثال:

If a vehicle floats on water, then it is a ship.

تكتب هذه الجملة اللغوية في البرولوغ بالشكل:

```
ship(X) :-
        vehicle(X),
        floats(X, water).
```

فمفردات اللغة الحية مثل: شيء ما، شخص ما، أي شيء، أي شخص، تفسر على أنها متحولات برولوغ. فيمكن إعادة صياغة الجملة السابقة بالشكل:

Something is a vehicle and floats on water, is a ship.

تقع (and) أحياناً في جواب الشرط، وبما أن رأس القاعدة لا يسمح بوجود علاقة منطقية ضمنه، لذلك يكرر جسم القاعدة كما يأتي:

If a vehicle floats on water and is gray, then it is a ship and military origin.

```
ship(X) :-
        vehicle(X),
        floats(X, water),
        gray(X).
```

```
origin(X, military) :-
        vehicle(X),
        floats(X, water),
        gray(X).
```

3. الاستعلام (Queries):

بعد قيام مترجم (Compiler) بترجمة (Compilation) برنامج البرولوغ، فإنه يصبح قابلاً للتنفيذ بكتابة الاستعلام المناسب في نافذة البرولوغ الخاصة بالأسئلة، والذي يشبه من ناحية البنية جسم القواعد، فهو يتألف من مجموعة من المعلنات تفصلها الفواصل العادية أو المنقوطة، وينتهي بنقطة. في حين تكون إشارة الجاهزية بالشكل (?-)، وهي تكتب عادة بشكل آلي في بداية كل سطر استعلام.

تكون نتيجة الاستعلام إما بالإيجاب أو النفي باستخدام (yes, no) أو باستخدام (true, false)، أو تكون معلومات أخرى. فمثلاً:

```
?- fine.
```

السؤال هنا مثلاً، هل الطقس رائع؟

الإجابة ستكون مختصرة بنعم أو لا. حيث يبحث البرولوغ في قاعدة معطيات البرنامج عن حقيقة مطابقة للاستعلام.

أما بكتابة:

```
?- is_bigger(elephant, tiger).
```

فالسؤال هنا، هل الفيل أكبر من النمر؟

أيضاً، ستكون الإجابة مختصرة بنعم أو لا.

يمكن أن يكون الاستعلام متضمناً أكثر من معلن، مع استخدام المتحولات:

```
?- small(X), green(X), beautiful(X).
```

عند وجود المتحول في السؤال، تكون الإجابة بالبحث عن المتحول الذي يحقق إجابة إيجابية، وإظهاره للمستخدم. في هذا السؤال، يقوم البرولوغ بالبحث عن اسم (term) يحقق ثلاث صفات في قاعدة المعطيات، وهي الحجم الصغير، واللون الأخضر، وجمال المنظر.

تكون الإجابة بإعطاء جميع الأسماء التي تحقق هذه الشروط، بشرط الاستمرار بالضغط على الفاصلة المنقوطة بعد كل إجابة، أو الضغط على زر المسافة (Space) في بعض البيئات البرمجية.

4. تنفيذ استعلام خلال ترجمة البرنامج:

يحتاج المبرمج أحياناً لإظهار بعض الرسائل التوجيهية للمستخدم، بشكل تلقائي عند ترجمة برنامج ما، لتنفيذ بعض الاستعلامات. كما أنه من غير العملي إعادة التصريح عن المعاملات الخاصة، كلما تطلب الأمر ذلك في نمط (Query Mode). فما هو الحل؟

الحل ببساطة، بوجود طريقة لكتابة الاستعلام في البرنامج أي في نمط (Program Mode)، وذلك بالشكل المبين:

```
:- write('Hello, have a beautiful day!').
```

وبالتالي، كلما تم ترجمة ملف البرنامج فإنه يقوم بتنفيذ ما بعد (:-). وهنا سيكتب على شاشة الحاسوب العبارة الآتية مباشرة:

```
?- Hello, have a beautiful day!
```

أو عند التنفيذ باستخدام المعلن (consult/1)، يظهر الشكل:

```
?- consult('C:/Users/test3.pl').
Hello, have a beautiful day!
% C:/Users/test3.pl compiled 0.00 sec, 7 clauses
true
```

7-6- مثال توضيحي (An Example):

لزيادة الإيضاح والتعرف على اللغة أكثر، يفضل تطبيق الشرح على مثال مباشر.

بفرض كتابة البرنامج الآتي في نافذة البرنامج:

```
a_kind_of(al_amal, ship).
a_kind_of(cham, ship).
part_of(al_amal, a_star_navy).
part_of(cham, a_star_navy).
part_of(a_star_navy, a_star_company).
a_kind_of(a_star_company, company).
color(ship, gray).
location(al_amal, p10n12e).
has(a_star_company, customer_service_system).
```

بعد عملية حفظ الملف بلاحقة (*.pl)، وترجمته من الخيار المتاح في البيئة البرمجية كالخيار (Compile Buffer) في بيئة (SWI-Prolog)، نكون قد حصلنا على ترجمة لقاعدة البيانات (Database)، وأصبحت جاهزة للإجابة عن الاستفسارات المطلوبة.

بالانتقال إلى نافذة الأوامر وكتابة الاستفسار الآتي:

```
?- part_of(cham, a_star_navy).
```

تبدأ عملية البحث من بداية قاعدة البيانات عن حقيقة مطابقة، وبما أنها موجودة فعلاً، فإن البرولوغ القياسي يعيد كلمة (yes) في نافذة الأوامر.

يوجد بيئات برمجة مختلفة للبرولوغ وبنسخ مختلفة أيضاً، حيث تستخدم بعضها كلمتي (true, false) بدلاً من (yes, no). سوف نعتمد في عملية الإظهار على بيئة (SWI-Prolog, Version 6.4.1) وهي نسخة سويدية تستخدم كلمتي (true, false).

بكتابة الاستفسار الآتي في نافذة الأوامر:

```
?- part_of(nawras, a_star_navy).
```

تبدأ عملية البحث في قاعدة البيانات عن حقيقة مطابقة، وبما أنها غير موجودة، يعيد البرولوغ كلمة (false) في نافذة الأوامر.

تترجم هذه الأسئلة على أنها تبدأ بصيغة السؤال "هل يوجد"، وتكون إجابة هذا النوع من الأسئلة بنعم أو لا فقط. لذلك ندعوها بأسئلة المطابقة . أما عند الرغبة بمعرفة معلومة مجهولة فيجب أن تكون صيغة السؤال "من يحقق"، وعندها يتم استخدام الأسئلة وحيدة المتحول، مثل:

```
?- part_of(cham, X).
```

فالسؤال هنا عن الأسطول الذي يحقق أن السفينة شام جزء منه. تستخدم المتحولات للسؤال عن المجهول، بمراعاة أن تبدأ بحرف كبير واحد على الأقل. تكون إجابة البرولوغ بعد عملية البحث من بداية قاعدة البيانات؛ هي:

```
X = a_star_navy.
```

لا يوجد شرط لمكان ورود المتحول، فمن الممكن كتابة السؤال:

```
?- part_of(X, a_star_navy).
```

للحصول على الجواب:

```
X = al_amal.
```

بما أن عملية البحث تبدأ من بداية قاعدة البيانات، فإن أول ورود لحقيقة تطابق السؤال الموجه، وبصرف النظر عن الوسيط الأول الذي وضع مكانه متحولاً هي التي تعتمد كإجابة أولى.

عند الرغبة بالحصول على ما تبقى من إجابات إن وجدت فإن البرولوغ يكون في حالة انتظار. فإذا ما قام المستخدم بالضغط على زر النقطة (.)، فهذا يعني إيقاف عملية البحث عن بقية الإجابات، والانتقال إلى تلقي أوامر جديدة. أما إذا قام المستخدم بالضغط على زر الفاصلة المنقوطة (;)، فهذا يعني الاستمرار في عملية البحث للحصول على إجابة أخرى. لذلك بإعادة كتابة السؤال الأخير وضغط زر الفاصلة المنقوطة بعد كل إجابة يظهر ما يأتي:

```
?- part_of(X, a_star_navy).
X = al_amal;
X = cham;
false.
```

إذن، يعطي البرولوغ أول حل للمسألة المطروحة في حال وجوده، ثم ينتظر أمراً من المستخدم. وبما أنه في السؤال الأخير لا يوجد سوى إجابتين، فإنه بعد الفاصلة المنقوطة الثانية لم يجد البرولوغ إجابة ثالثة فأعطى (false).

عند ورود أكثر من متحول في السؤال تصبح عملية البحث أوسع وذات خيارات أكثر. يرد البرولوغ بمجموعات من المتحولات التي تمثل حلولاً مختلفة للمسألة المطروحة عموماً، وكمثال على حالة تعدد المتحولات:

```
?- part_of(X, Y).
X = al_amal,
Y = a_star_navy;
X = cham,
Y = a_star_navy;
X = a_star_navy,
Y = a_star_company;
false.
```

تفصل إشارة الفاصلة العادية بين قيم مجموعات المتحولات وكأنها تعني أداة العطف (و)، وهي توضع من قبل البرنامج. أما الفاصلة المنقوطة فتدخل من قبل المستخدم لتفصل بين الإجابات المختلفة.

لا تقتصر عملية الاستفسار على الأسئلة وحيدة الشرط، بل يمكن طرح أسئلة متعددة الشروط. عند الرغبة بمعرفة لون سفينة شام فإن المستخدم يكتب:

```
?- color(cham, C).
false.
```

الإجابة بالنفي منطقية، لعدم وجود حقيقة مطابقة للسؤال. أما بكتابة السؤال بالشكل الآتي:

```
?- a_kind_of(cham, T), color(T, C).
T = ship,
C = gray.
```

يترجم السؤال بالشكل اللغوي الآتي:

إلى أي نوع تتبع سفينة شام، وما هو لون هذا الشيء؟

إذن، الفاصلة (, Comma) بين التعبيرين في السؤال تعني أداة العطف (و) أيضاً، ولكن هنا تستخدم كعملية منطقية (and)، توجب تحقق كلا التعبيرين سوية. تكون الفاصلة بين هدفين جزئيين (Subgoal) في الاستعلام أو في جسم القاعدة وتعني علاقة ضم واتحاد (Conjunction). لذلك يقوم البرولوغ بالبحث عن إجابة التعبير الأول، وعندما يجد حلاً مقبولاً، ينتقل للبحث عن إجابة للتعبير الثاني، بمراعاة تغيير قيمة المتحول من إجابة التعبير الأول في التعبير الثاني. كمثال آخر نكتب:

```
?- part_of(cham, X), part_of(X, Y).
X = a_star_navy,
Y = a_star_company.
```

بما أنه يوجد عملية (و) المنطقية، فلا بد من وجود عملية (أو) المنطقية، التي تمثل باستخدام الفاصلة المنقوطة (; Semicolon)، وتعني علاقة فصل (Disjunction) أو بشكل منطقي علاقة (or)، كما هو موضح في المثال المبين:

```
?- color(cham, C); color(ship, C).
C = gray.
```

يبحث البرولوغ عن جميع إجابات التعبير الأول ويظهرها إن طلب المستخدم ذلك، وبعد الانتهاء من جميع الإجابات الممكنة، ينتقل للبحث عن جميع إجابات التعبير الثاني ويظهرها إن طلب المستخدم ذلك، كما هو مبين أدناه:

```
?- a_kind_of(X, ship); a_kind_of(Y, company).
```

X = al_amal;

X = cham;

Y = a_star_company.

يجب الانتباه إلى أنه عند القول بأن البرولوغ يبحث عن جميع إجابات التعبير الإعلاني، فهذا لا يعني أنه يقوم بعملية البحث كاملة، ويخزن النتائج، ثم يظهر الإجابات بشكل متتال حسب طلب المستخدم. فالحقيقة أن البرولوغ يبدأ بعملية البحث من بداية البرنامج، وعندما يجد أول إجابة فإنه يظهرها للمستخدم وينتظر في المكان الذي وصل إليه في البرنامج. فإذا طلب المستخدم إجابة أخرى، تابع عملية البحث من النقطة التي وصل إليها.

عند وجود شروط متداخلة، فيمكن استخدام الأقواس للفصل بين التعابير الإعلانية، وهي ذات الأولوية أو الأسبقية الأعلى، وتأتي بعدها أولوية الفاصلة المنقوطة ومن ثم أولوية الفاصلة العادية.

?- part_of(cham, X); (part_of(cham, Y), part_of(Y, Z)).

X = a_star_navy ;

Y = a_star_navy,

Z = a_star_company.

إذا وجد في البرنامج علاقتان متطابقتان بالاسم، فإن البرنامج سينفذ إحداهما. في البداية تنفذ العلاقة الواردة أولاً، وتنفذ الثانية فقط في حال الإجابة السلبية للعلاقة الأولى، أو عندما يطلب المستخدم حلاً آخر، وهذا ما يدعى بعلاقة الفصل (Disjunction).

وللسهولة فقد وجدت الفاصلة المنقوطة لعدم تكرار العلاقة.

المثال الآتي يوضح ذلك:

```
parent(X, Y) :-
        father(X, Y).
parent(X, Y) :-
        mother(X, Y).
```

بالطبع، سيتم اختبار إحدى العلاقتين، وعند تحقق واحدة لا تختبر الثانية، إلا إذا طلب المستخدم ذلك. ويمكن للاختصار استخدام الفاصلة المنقوطة بالشكل:

```
parent(X, Y) :-
        father(X, Y);
        mother(X, Y).
```

من المفضل في البرولوغ عدم استخدام عملية (أو) المنطقية أو التقليل قدر المستطاع منها، والاستعاضة عنها بتكرار اسم العلاقة.

إن العمليات المنطقية الأساسية هي (and, or, not)، والتي تنفذ باستخدام الفاصلة من أجل (and)، والفاصلة المنقوطة من أجل (or). أما من أجل (not)، فتستخدم الكلمة نفسها. إن عملية النفي الإعلانية في البرولوغ من أعقد العمليات المنطقية. لذلك سوف نتعرف عليها مبدئياً من خلال مثال بسيط تاركين بعض التفاصيل عنها لفقرات لاحقة.

تتحقق عملية (not) في برنامج البرولوغ عندما يخفق السؤال ضمن قوسيها، وتخفق عندما يتحقق السؤال.

عند طلب الاستفسار الآتي:

```
?- not(color(cham, green)).
```

تكون الإجابة (true)، فهو يعني الاستفسار عن عدم وجود حقيقة ما، وهنا الاستفسار عن عدم وجود حقيقة هي أن لون سفينة شام هو الأخضر. فعملية (not) في البرولوغ لا تعني عملية منطقية، بل تعني وجود أو عدم وجود الحقيقة المطلوبة.

للسؤال مثلاً عن أحد أنواع الأشياء غير الرمادية، نكتب:

```
?- a_kind_of(X, Y), not(color(Y, gray)).
X = a_star_company,
Y = company.
```

إن الإجابة عن الاستفسارات السابقة، ليست سوى معالجة بسيطة لقاعدة المعطيات، ولا يتضح من خلالها أي ذكاء. حتى يتمتع برنامج البرولوغ بالذكاء، يجب إدخال بعض القواعد إلى البرنامج.

بإضافة القاعدة الآتية إلى نهاية البرنامج السابق نجد:

```
gray_cham :-
        part_of(cham, X),
        color(X, gray).
```

تتحقق القاعدة عندما يتحقق جسم القاعدة وتكون إجابته (true)، وكأن (gray_cham) اسم معلن لحقيقة جديدة تكافئ السؤال في الطرف الأيمن من القاعدة.

يعامل المتحول (X) كمتحول محلي (Local Variable)، وكذلك أي متحول في الطرف الأيمن.

باستدعاء اسم القاعدة في نافذة الاستفسار:

```
?- gray_cham.
```

يترجم الاستفسار السابق لغوياً على أنه هل من شيء ما لونه رمادي وسفينة شام جزء منه؟. وتكون الإجابة بالنفي أي (false)، نظراً لعدم تحقق الطرف الأيمن.

لم يتضمن اسم معلن القاعدة في الرأس أي متحول، فيكون الجواب بالنفي أو الإيجاب فقط. أما للحصول على نتيجة ذات فائدة عملية فيجب تضمين أسماء متحولات في رأس القاعدة. فللسؤال عن لون شيء ما وبشرط أن تكون سفينة شام جزء من هذا الشيء نكتب في جزء البرنامج:

```
color_cham(X) :-
        a_kind_of(cham, Y),
        color(Y, X).
```

باستدعاء اسم القاعدة في نافذة الاستفسار مع المتحول (Z):

```
?- color_cham(Z).
Z = gray.
```

تكون الإجابة بإظهار قيمة المتحول (Z)، الذي يحقق أن يكون جسم القاعدة صحيحاً. ليس من الضروري طبعاً، استدعاء اسم المعلن مع متحولاته بأسماء مطابقة لرأس القاعدة في البرنامج. فعند استخدام المتحول (Z) في الاستفسار، يقوم البرولوغ بوضعه مكان المتحول (X) في البرنامج. يدعى المتحول (Z)، متحولاً وسيطاً (Parameter Variable)، حيث يسترد قيمته بعد تنفيذ القاعدة.

يمكن استخدام عدة متحولات في القاعدة، كإضافة القاعدة الآتية إلى البرنامج:

```
color_object(X, C) :-
        a_kind_of(X, Y),
        color(Y, C).
```

باستدعاء القاعدة مع المتحولات:

```
?- color_object(M, N).
M = al_amal,
N = gray;
M = cham,
N = gray;
false.
```

يوجد إجابتان مختلفتان للاستفسار، وكل إجابة تتضمن قيم جميع المتحولات الخاصة بهذه الإجابة.

ينظر للمتحولات المحلية كتكميم وجودي (Existential)، أي يوجد (X) بحيث أن شيئاً ما محققاً. وينظر للمتحولات الوسيطة كتكميم عام (Universal)، أي لكل (X) فإن شيئاً ما محققاً. ولا يوجد متحولات عامة (Global Variable)، ولكن يمكن تمثيلها كحقائق.

تكتب الحقائق والقواعد في البرولوغ بشكل مرتب بحيث يفضل أن يبدأ البرنامج بالحقائق ثم تتبعها القواعد. كما أن يفضل أن ترد الحقائق المتطابقة باسم المعلن بشكل متتابع كي لا يعطي المترجم رسالة تنبيه عند ترجمة الملف، ولزيادة تنظيم البرنامج. وقد تقصدنا إيراد هذه الحالة في المثال السابق، حيث وضعت الحقيقة (a_kind_of(a_star_company, company).) بشكل مستقل وبعيد عن بقية الحقائق المتطابقة باسم المعلن.

ذكرنا سابقاً أنه يفضل عدم استخدام عملية (أو) المنطقية في البرولوغ. ولنأخذ المثال الآتي:

بإضافة القاعدة الآتية إلى البرنامج:

```
color_object(X, C) :-
        color(X, C);
        (part_of(X, Y), color(Y, C));
        (part_of(X, Y), part_of(Y, Z), color(Z, C)).
```

فمن الملاحظ صعوبة قراءة القاعدة وفهمها، لذلك يمكن الاستعاضة عنها بإعادة كتابتها دون عملية (أو) وبالمعنى نفسه، ولكن بشكل أوضح بالشكل:

```
color_object(X, C) :-
        color(X, C).
color_object(X, C) :-
        part_of(X, Y),
        color(Y, C).
color_object(X, C) :-
        part_of(X, Y),
        part_of(Y, Z),
        color(Z, C).
```

عند السؤال عن المعلن (color_object)، فإن البرولوغ يبدأ بفحص القواعد بالترتيب الوارد ضمن البرنامج معطياً الإجابة الأولى. من المحتمل أن يكتفي المستخدم بإجابة واحدة تمثل حلاً للمسألة المطروحة، دون الالتفات أو الاهتمام ببقية الحلول. بما أن الاستخدام العملي للبرولوغ يكون للاستفسار من

قاعدة معطيات ضخمة، وهذا ما يؤثر على الوقت المستغرق في إعطاء الإجابة. إذن، كتابة البرنامج باحترافية هو أمر مطلوب، فعند كتابة القاعدة (color_object) بالأسلوب الأول فهذا يتطلب وقتاً في الإجابة، أما كتابتها بالأسلوب الثاني فإن وقت الإجابة يكون أقصر، خاصة بكتابة القواعد (color_object) بشكل مرتب من الأبسط نحو الأعقد. عند طلب الاستفسار فإن البرولوغ يبحث من بداية البرنامج، وعندما يفحص أول (color_object) بجسم يحوي معلناً واحداً، فهذا سيتطلب وقتاً أقصر من فحص ثاني (color_object) بجسم يحوي معلنين اثنين، وهذا يتطلب أيضاً وقتاً أقصر من فحص ثالث (color_object) بجسم يحوي ثلاثة معلنات.

7-7- أساسيات التعقب الخلفي (Backtracking):

1. تعريف التعقب الخلفي:

يفيد التعرف على عملية التعقب الخلفي في معرفة منهجية البرولوغ في الحصول على إجابات الأسئلة أو الاستفسارات الموجهة له. في حالة التعابير الإعلانية المرتبطة فيما بينها بعملية (و) المنطقية، فإن البرولوغ يبدأ من أقصى اليسار وحتى اليمين، وكلما تحقق تعبير إعلاني، انتقل إلى التعبير الذي يليه. عند الإخفاق في أي تعبير إعلاني فإنه يعود إلى التعبير الذي يسبقه ويجرب قيماً أخرى أو إجابات أخرى، إن وجدت. بمجرد حصوله على إجابة صحيحة للتعبير الحالي فإنه يعاود الكرة بالبحث عن حلول للتعبير اللاحق، آخذاً بالاعتبار ما تم تحديثه في المرحلة الأخيرة. عند وصول البرولوغ إلى التعبير الأخير وإيجاد إجابة مقبولة له، فإنه يظهر إجابة الاستفسار الموجه له كاملة. ينتظر البرولوغ أحياناً أمراً من المستخدم، فإذا أراد حلاً آخراً، قام بالضغط على زر الفاصلة المنقوطة،

ليقوم البرولوغ بالعودة إلى النقطة التي توقف عندها، ومتابعة عملية البحث عن بقية الحلول، حتى يصل إلى إجابة أو حل آخر للمسألة المطروحة، فيظهره وينتظر أمر المستخدم. عندما يكتفي المستخدم بالإجابات يضغط على مفتاح النقطة ليتوقف البرولوغ عن متابعة البحث وينتقل إلى إشارة الجاهزية لتلقي استفسارات جديدة. بعض المسائل لها عدة حلول وبعضها له عشرات أو آلاف الحلول المقبولة. والمستخدم لا يهمه دائماً الحصول على كامل الحلول، فيكتفي بأي حل مقبول، لذلك فالبرولوغ يوفر وقت عملية البحث عن كامل الحلول بإظهارها واحداً تلو الآخر، بناء على رغبة المستخدم. على سبيل المثال، مسألة الوزراء الثمانية في الشطرنج لها (92) حلاً مختلفاً.

لنكتب الاستفسار الآتي، لمثال فقرة المثال التوضيحي:

```
?- part_of(X, Y), has(Y, customer_service_system).
```

يوجد في قاعدة معطيات البرنامج ثلاث حقائق للمعلن (part_of)، وحقيقة واحدة للمعلن (has). يبدأ مترجم البرولوغ من السؤال الفرعي الأول، أي من التعبير الإعلاني الأول، ليطابق مع السطر الأول في البرنامج، فالثاني، فالثالث. يوجد تطابق بأسماء المعلنات في السطر الثالث مع تطابق عدد الوسطاء. تتم عملية مطابقة للوسطاء، فإذا كان أحد الوسطاء متحولاً، يتم إسناد قيمة الوسيط غير المتحول له فتصبح القيم كما يأتي:

```
X = al_amal,
Y = a_star_navy.
```

لا تظهر هذه القيم للمستخدم حالياً، لأن العلاقة مع التعبير الثاني هي (و) المنطقية، فيتطلب الأمر تحقق هذه القيم في التعبير الإعلاني الثاني أيضاً.

إذن، ينتقل مترجم البرولوغ إلى اختبار تحقق التعبير الإعلاني الثاني مع تمرير قيمة المتحول (Y) له. يبدأ البرولوغ بالبحث من بداية البرنامج عن تطابق للحقيقة الآتية:

has(a_star_navy, customer_service_system).

لكنه لا يجد حقيقة مطابقة لها.

في هذه اللحظة تحديداً تبدأ عملية التعقب الخلفي، حيث يعود البرنامج إلى التعبير السابق مباشرة ليبحث عن حل آخر له انطلاقاً من النقطة التي توقف عندها. يتابع البرولوغ من السطر الرابع فيجد تطابقاً بأسماء المعلنات مع تطابق عدد الوسطاء. تتم عملية مطابقة للوسطاء من جديد، فتصبح القيم الجديدة كما يأتي:

X = cham,

Y = a_star_navy.

أيضاً، ينتقل مترجم البرولوغ إلى اختبار تحقق التعبير الإعلاني الثاني مع تمرير القيمة الجديدة للمتحول (Y) له. يبدأ البرولوغ بالبحث من بداية البرنامج عن تطابق للحقيقة الآتية:

has(a_star_navy, customer_service_system).

لكنه لا يجد حقيقة مطابقة لها.

نلاحظ أن البرولوغ أعاد تكرار الخطوات نفسها، ولم يتذكر أن هذه الخطوات مطابقة للخطوات السابقة، إذ لم تتغير قيمة المتحول (Y)، وقيمة المتحول (X) لا تدخل في التعبير الثاني. سنتعرف لاحقاً على كيفية برمجة البرولوغ، بحيث يتذكر الخطوات المكررة، ويتجاوزها لتزيد سرعة الاستجابة بشكل ملحوظ.

يعود البرنامج للتعقب الخلفي أيضاً، ويبدأ من السطر الخامس فيجد تطابقاً بأسماء المعلنات مع تطابق عدد الوسطاء. تتم عملية مطابقة للوسطاء من جديد، فتصبح القيم الجديدة كما يأتي:

X = a_star_navy,

Y = a_star_company.

أيضاً، ينتقل مترجم البرولوغ إلى اختبار تحقق التعبير الإعلاني الثاني مع تمرير القيمة الجديدة للمتحول (Y) له. يبدأ البرولوغ بالبحث من بداية البرنامج عن تطابق للحقيقة الآتية:

has(a_star_company, customer_service_system).

فيجدها في السطر التاسع والأخير من البرنامج.

هنا، يقوم البرنامج بإعطاء النتيجة التي تمثل حلاً للمسألة المطروحة أو الاستفسار المطلوب، وهي عبارة عن قيم المتحولات التي حققت صحة التساؤل الإعلاني بالشكل المبين:

X = a_star_navy,

Y = a_star_company.

لا يوجد إجابات ممكنة إضافية، لذلك تتوقف عملية البحث، وينتقل البرولوغ إلى إشارة الجاهزية لتلقي استفسارات جديدة.

لو طرح الاستفسار السابق بالشكل الآتي:

?- has(Y, customer_service_system), part_of(X, Y).

لما احتاج الحصول على النتيجة السابقة نفسها وقتاً طويلاً، لأن البرولوغ سيتمكن من المرحلة الأولى في عملية البحث من الوصول إلى النتيجة، ودون الحاجة لعملية التعقب الخلفي التي تستهلك زمناً طويلاً نسبياً.

يقوم البرولوغ عموماً بتجميع جميع الحقائق المتطابقة باسم المعلن وعدد الوسطاء، ويعطي لكل منها دليلاً. وبالتالي تفهرس الحقائق، وعند البحث عن حقيقة ما، فإن مترجم البرولوغ يستعين بهذا الفهرس كي يبحث فقط عن هذه الحقيقة.

عندما تكون العلاقة المنطقية الرابطة بين التعابير الإعلانية في جسم القاعدة هي علاقة (أو)، فإن البرولوغ يتجاهل بقية التعابير الإعلانية عند تحقق التعبير الأول ويعطي النتيجة مباشرة.

2. مثال عن التعقب الخلفي:

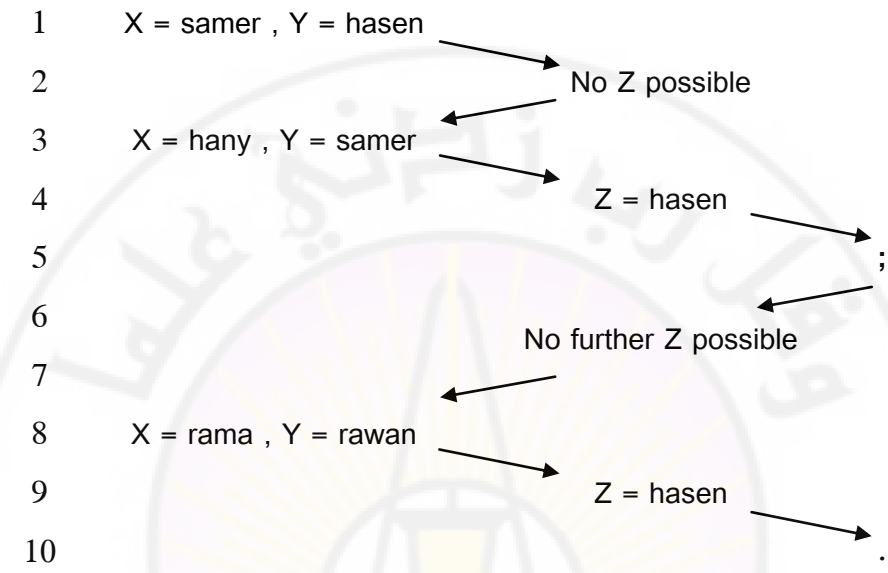
لتكن قاعدة المعطيات المبينة:

```
boss(samer, hasen).
boss(hany, samer).
boss(rama, rawan).
boss(rawan, hasen).
```

نبين فيما يأتي خطوات تنفيذ عملية التعقب الخلفي بشكل تخطيطي في الشكل(7-2) بكتابة الاستفسار الآتي:

```
?- boss(X, Y), boss(Y, Z).
```

الخطوة	التعبير الإعلاني الأول	التعبير الإعلاني الثاني



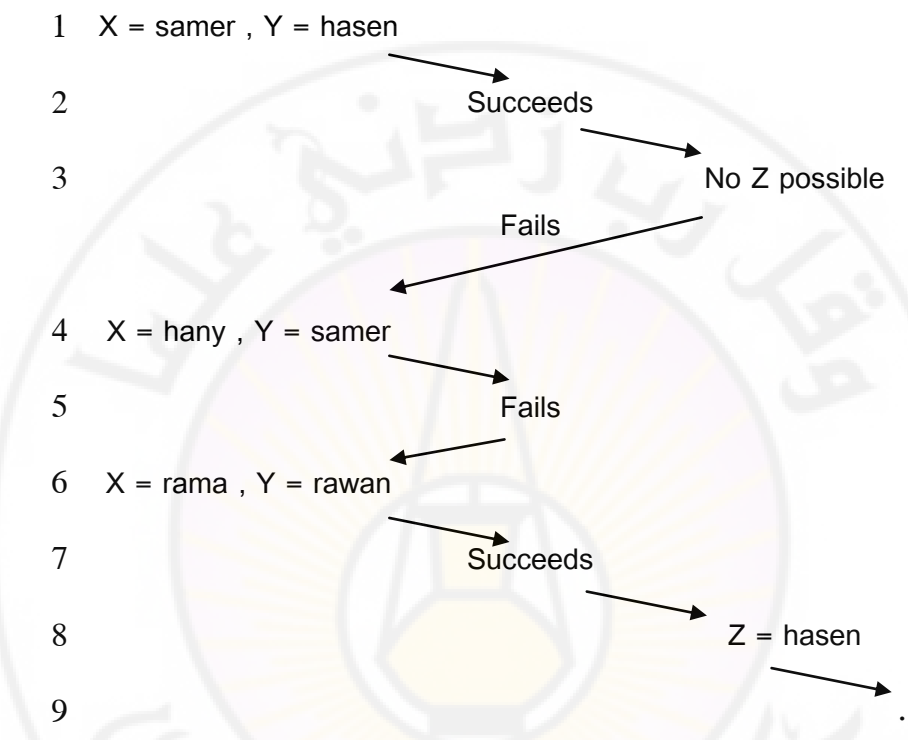
الشكل(7-2) تنفيذ عملية التعقب الخلفي بشكل تخطيطي

أما لدراسة التعقب الخلفي مع عملية (not) المنطقية فيمكن توضيحها في الشكل(7-3) بكتابة الاستفسار الآتي:

?- boss(X, Y), not(boss(X, samer)), boss(Y, Z).

وهو يعني إيجاد تتابع الإدارة لثلاثة مستويات، من المدير إلى رئيس العمال فالعاملين، مع استثناء حالة كون رئيس العمال هو (samer).

الخطوة	التعبير الإعلاني الأول	التعبير الإعلاني الثاني	التعبير الإعلاني الثاني



الشكل(7-3) تنفيذ عملية التعقب الخلفي بشكل تخطيطي مع (not)

من الواضح أنه عند الانتقال من الخطوة (3) إلى الخطوة (4) لم تؤخذ عملية التعقب الخلفي بالاعتبار للتعبير الإعلاني الثاني. يفسر هذا الأمر بأن عملية (not) المنطقية تفشل دائماً في التعقب الخلفي.

بشكل مشابه، لكنه أوضح، يمكن كتابة الاستفسار الآتي:

?- boss(X, Y), boss(Y, Z) , not(boss(X, samer)).

وهو يعطي نتيجة مطابقة للاستفسار السابق.

لكن عند بدء الاستفسار بالشكل الآتي:

```
?- not(boss(X, samer)), boss(X, Y), boss(Y, Z).
```

فإن العملية لا تنفذ، لأن قيمة المتحول (X) في عملية (not) المنطقية ما زالت غير مرتبطة بأية قيمة.

3. التعقب الخلفي مع القواعد (Backtracking with Rules):

تمتاز البرولوغ بإمكانية تأجيل معالجة القواعد، وهو ما يسمى بالربط المؤجل (Postponed Binding)، فمثلاً:

```
?- color_object(cham, C).
```

الوسيط الأول يمثل الدخل وهو مرتبط كونه (Atom). أما الوسيط الثاني فيمثل الخرج وهو غير مرتبط، فإذا وجدت حقيقة في قاعدة البيانات؛ مثل:

```
color_object(cham, red).
```

عندئذ يرتبط المتحول (C) مع (red).

أما إذا لم توجد حقائق (color_object) بل وجدت قواعد (color_object)، أو كانت هذه القاعدة في بداية قاعدة البيانات، فتؤجل عملية الربط.

```
color_object(X, C) :-
        part_of(X, Y),
        color_object(Y, C).
```

وربما تطلب الأمر عدة استدعاءات متكررة للمعلن (color_object)، حتى تتم عملية الربط، وهذا ما يعني الكلفة الزمنية العالية.

أحياناً لا يمكن ربط المتحولات مع أي قيمة؛ مثل:

```
recommended(X, J) :-
        not(bad_report(X)).
```

حيث يرشح (X) للعمل (J) إذا لم يكن عليه تقرير سلبي، فهنا (J) لا ترتبط مع أي شيء. وبالتالي فإن الربط يحدث فقط عند الإجابة الضرورية بشكل فعلي.

من الصعب اقتفاء التعقب الخلفي للقواعد، ولتوضيح ذلك لنفرض المثال الآتي:

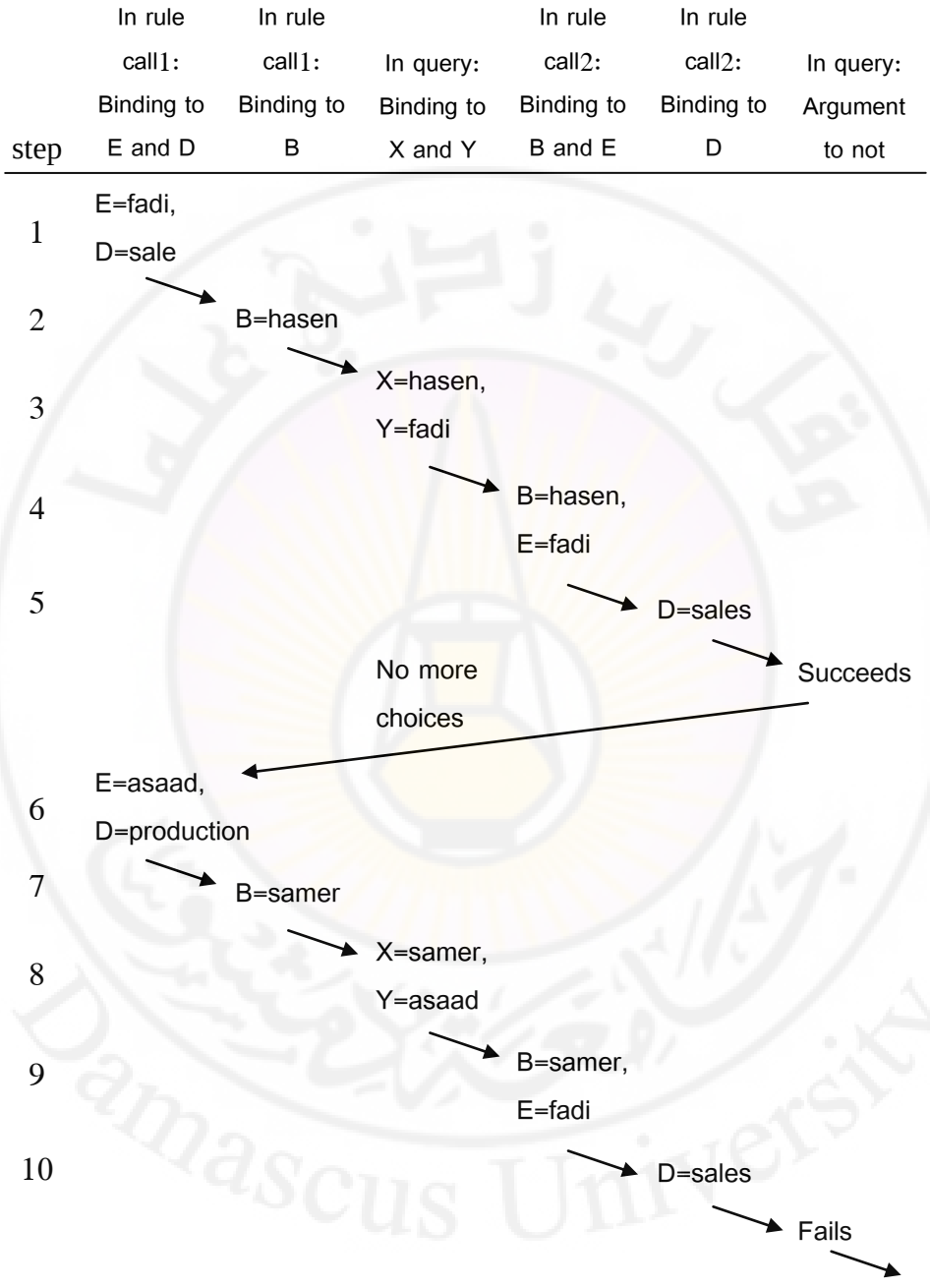
```
department(fadi, sales).
department(asaad, production).
manager(hasen, sales).
manager(samer, production).
boss(B, E) :-
        department(E, D),
        manager(B, D).
```

القسم مكون من عمال ورؤسائهم، ويكون (B) رئيس (E) إذا كان (E) يعمل في القسم (D) الذي يرأسه (B).

لنبين مراحل تنفيذ التعقب الخلفي مع القواعد للتساؤل الآتي:

```
?- boss(X, Y), not(boss(X, fadi)).
```

يبين الشكل(7-4) عملية التعقب الخلفي بشكل تخطيطي مع (rule).



الشكل(7-4) تنفيذ عملية التعقب الخلفي بشكل تخطيطي مع (rule)

7-8- مثال متكامل (An Example):

قبل متابعة الخوض في مواضيع أخرى في لغة البرولوغ نطرح المثال العام المتكامل الآتي مع إعادة شرح بعض النقاط. يبين المثال كيفية تعريف الحقائق والقواعد وكيفية الاستعلام، وذلك بهدف البدء بالتأقلم مع منهجية اللغات الوصفية، ودون الخوض بالتفاصيل. من الواضح أنه يجب شرح المسألة المطروحة بشكل جيد وكامل بعبارات وصفية واضحة، وبمراعاة بعض القواعد المتبعة في أثناء كتابة برنامج البرولوغ. يتم البدء بوصف المسألة باستخدام الحقائق ، ولنبدأ بحقائق بسيطة تصف أحجام بعض الحيوانات بشكل نسبي، للانتقال بعد ذلك إلى كيفية الحصول على معلومات جديدة لم تكتب بشكل مباشر، ولكنها مستنتجة من الحقائق المعروفة.

نكتب في المكان المخصص للبرنامج (Program Mode) ما يأتي:

```
bigger(elephant, horse).
```

هذه حقيقة صحيحة وبديهية، وهي تترجم بالشكل الذي يرغبه المبرمج ، فهنا تقول الحقيقة أن الفيل أكبر من الحصان.

يمكن كتابة عدة حقائق بشكل مشابه، وبالنمط المستخدم نفسه، في المكان المخصص للبرنامج، كما هو مبين:

```
bigger(elephant, horse).
bigger(horse, tiger).
bigger(tiger, sheep).
bigger(tiger, monkey).
```

وهنا لا بد من الإشارة إلى أن المبرمج اعتمد هنا على أن الكائن الحي ذي الحجم الأكبر يكتب أولاً، بحيث تقرأ الحقيقة بالترتيب نفسه، أي الكائن الأول

أكبر من الكائن الثاني. ولا يوجد مانع من القراءة بشكل معاكس إن كان هذا صحيحاً عملياً ، ولكن المهم المتابعة بالصيغة نفسها. فلا يجوز أن نعتمد مرة الكائن الأول هو الأكبر وفي عبارة أخرى نكتب الكائن الثاني هو الأكبر، فالبرولوغ لا يعرف هذه الحقائق على أنها صحيحة في الواقع أم لا، بل يتعرف عليها كعلاقات، وهذا واضح ، فالحاسوب لا يفهم على أن الفيل كائن حي كبير الحجم وله خرطوم وأنياب، بل يتعرف عليه كاسم ثابت فقط، وتربطه علاقة اسمها (bigger) مع اسم ثابت آخر. من الملاحظ أنه تكرر اسم الحقيقة عدة مرات، بالصيغة نفسها، لكن بتغيير الأسماء ، وانتهى كل سطر بنقطة، كما استخدمت أسماء ذات معنى، وجميع هذه الأسماء سواء المعلنات أم الوسطاء غير معرفة مسبقاً في البرولوغ.

تشكل الأسطر السابقة برنامجاً في لغة البرولوغ، وما على المبرمج سوى حفظه كملف وترجمته من القوائم الموجودة في البيئة البرمجية. بعد ترجمة الملف، وتلقي رسالة بعدم وجود أخطاء محرفية أو نحوية في البرنامج، فإنه يمكن البدء بالاستعلام وطرح الأسئلة على البرولوغ من (Query Mode) والتي تبدأ بإشارة استفهام وشرطة، وينتهي سطر الاستعلام بنقطة حتماً، كما يأتي:

```
?- bigger(tiger, sheep).
true.
```

بمجرد إنهاء كتابة السطر وضغط زر الإدخال (Enter) فإن البرولوغ يرد بأن الاستعلام صحيح، بكتابة (true).

لقد كتب استعلام بصيغة الحقيقة نفسها، وحتى بالأسماء نفسها، وهي تعني ضمن نمط الاستعلام سؤالاً وليس حقيقة، وهذا مشابه تماماً لطريقتنا في

التحدث. فنحن نقول أحياناً خلال حديثنا العادي مع طالب حواسيب مجتهد الجملة الآتية:

" تفوقت وتميزت خلال مناقشة مشروعك ". هذه الجملة تفهم كحقيقة وكإخبار من قبلنا لهذا الطالب بأنه تفوق وتميز، وذلك عندما نقولها بلهجة مناسبة. ولكن الجملة نفسها تفهم كسؤال ، عندما نقولها بلهجة استعلام رافعين حاجبينا ومعبرين عن رغبتنا بالاستعلام، عما إذا كان هذا الطالب قد تفوق وتميز، ليجيبنا عندها بكلمة نعم أو لا. بالطبع البرولوغ وعند سؤاله فإنه يذهب ويبحث في البرنامج أي قاعدة معطياته عن تطابق هذا السؤال مع حقيقة لديه، ليعيد الإجابة بالإيجاب عند التطابق وبالنفي عند عدم التطابق. فهنا نلاحظ أننا حصلنا على إجابة نعم لأن قاعدة المعطيات فيها حقيقة مطابقة للسؤال ، وهي أن النمر أكبر من الخروف.

في حال كان السؤال عن حقيقة غير مدرجة ضمن قاعدة المعطيات، كأن نسأل فيما إذا كان القرد أكبر من الفيل، عندها ستكون الإجابة بالنفي كما هو مبين:

```
?- bigger(monkey, elephant).
false.
```

طبعاً، الإجابة متوقعة لأنها صحيحة بالواقع فالقرد ليس أكبر من الفيل. أيضاً فإن قاعدة المعطيات لم تتضمن هكذا حقيقة.

ولكن ماذا نتوقع أن تكون إجابة البرولوغ عندما نسأله فيما إذا كان الفيل أكبر من القرد؟

```
?- bigger(elephant, monkey).
false.
```

من ناحية نقول : إن الإجابة بالنفي، مناقضة للمنطق الواقعي، لأن الفيل أكبر من القرد، وكان من المفترض أن تكون الإجابة عن السؤال بنعم. ومن ناحية أخرى نقول : إن هذه الحقيقة لم ترد في قاعدة المعطيات لذلك كانت الإجابة بالنفي.

بمراجعة البرنامج فإننا نلاحظ وجود حقائق متتابعة، فالفيل أكبر من الحصان، والحصان أكبر من النمر، والنمر أكبر من القرد، وهذا يعني منطقياً أن الفيل سيكون أكبر من القرد، فعلاقة الحجم رياضياً علاقة متعدية (Transitive)، ولكنها في البرولوغ غير معروفة، فهي علاقة اسمها (bigger) فقط، فكيف لنا أن نعرفها على أنها متعدية أو متصفة بأي صفة أخرى. فإذا كان لزاماً علينا كتابة جميع الحقائق في قاعدة المعطيات، عندها أين الذكاء!؟. إذن، لا بد من وجود طريقة تقلل من كتابة الحقائق التي يمكن استنتاجها بعلاقات ما.

لدينا هنا عدة حقائق إضافية يمكن استنتاجها من الحقائق الأربع السابقة ذهنياً. فيمكن استنتاج أن الفيل أكبر من النمر والخروف والقرد، وأن الحصان أكبر من الخروف والقرد، أي خمس حقائق إضافية مستنتجة ذهنياً، ولا يستطيع البرولوغ الذكي إيجادها، إلا إذا كتبت كحقائق!. إذن، ما الحل؟.

إن الحل هو باستخدام القواعد (Rules)، والتي هي عبارة عن علاقات مشروطة. لنعرف قاعدة كون كائن حي أكبر من آخر بشكل وصفي كما يأتي:

يكون (X) أكبر من (Y) إذا كان لدينا حقيقة مباشرة في قاعدة المعطيات تنص على ذلك، أو عندما يكون لدينا حقيقة مباشرة في قاعدة المعطيات تنص على أن (X) أكبر من (Z)، وحقيقة مباشرة أخرى في قاعدة المعطيات تنص على أن (Z) أكبر من (Y)، عندها يجب أن نستنتج أن (X) أكبر من (Y). ولكتابة

هذه القاعدة التي سندعوها (is_bigger) فإننا نضيف الأسطر الآتية إلى قاعدة المعطيات:

```
is_bigger(X, Y) :-
        bigger(X, Y).
is_bigger(X, Y) :-
        bigger(X, Z),
        bigger(Z, Y).
```

تترجم القاعدة الأولى بالشكل:

يكون (X) أكبر من (Y) فيما إذا وجدت حقيقة مباشرة في قاعدة المعطيات تنص على ذلك اسمها (bigger). وبالتالي فإشارة (:-) تعني (فيما إذا كان). تم البدء بتسمية القاعدة بفعل الكون (is) لفهم المعنى فقط، حيث يمكن التسمية بأي شكل آخر موافق لقواعد التسمية في البرولوغ.

ولكن الأسماء (X) و(Y) لم يردا في قاعدة المعطيات فماذا يعنيان ؟.

إن (X) و(Y) عبارة عن متحولات وليست أسماء ثابتة، حيث تميز المتحولات باستخدام حرف كبير في بداية الاسم إن كان متعدد الأحرف، أو حرف كبير واحد للبرامج البسيطة.

إن الاكتفاء بالقاعدة الأولى ليس ذا جدوى، ولن يقدم شيئاً جديداً. أما إذا نظرنا للقاعدة الثانية، فعندها سوف تتبين لنا فائدة العلاقات.

تترجم القاعدة الثانية بالشكل:

يكون (X) أكبر من (Y) فيما إذا وجدت حقيقة مباشرة في قاعدة المعطيات تنص على أن (X) أكبر من شيء ما هو (Z)، وأيضاً وجدت حقيقة

مباشرة في قاعدة المعطيات تنص على أن (Z) أكبر من (Y)، فإشارة الفاصلة (comma ,) تعني علاقة (و) المنطقية.

إذن، أصبح بإمكاننا التعرف على الحيوان الأكبر ولو لم يرد صراحة ضمن الحقائق، ولكن كعلاقة متعدية لمستو واحد فقط مثل:

```
?- is_bigger(elephant, tiger).
true.
```

أما لأكثر من مستوي فيجب تكرار عدة قواعد وبمستويات تعد متزايدة. بالطبع هذا الحل غير عملي، لذلك يمكن إضافة القاعدة الآتية لحل هذه المشكلة:

```
is_bigger(X, Y) :-
        bigger(X, Z),
        is_bigger(Z, Y).
```

تترجم القاعدة الثالثة بالشكل:

يكون (X) أكبر من (Y) فيما إذا وجد متحول آخر اسمه (Z) ووجدت حقيقة مباشرة في قاعدة المعطيات اسمها (bigger) تنص على أن (X) أكبر من (Z) ووجدت علاقة اختبار الأكبر بين (Z) و(Y). وبالتالي فإشارة الفاصلة (comma ,) تعني (و).

نلاحظ أنه بإمكاننا تكرار اسم معلن العلاقة أكثر من مرة وبأكثر من شكل. ونلاحظ أننا استخدمنا اسم معلن العلاقة ضمن العلاقة.

قبل متابعة الشرح، لنكتب الاستعلام الآتي بعد ترجمة البرنامج:

```
?- is_bigger(elephant, monkey).
true.
```

لقد تغيرت الإجابة، فما الذي حدث !؟

أول شيء تغير هو أننا قمنا باستدعاء علاقة، وليس حقيقة كما سبق. يبدأ البرولوغ لتنفيذ هذا الاستعلام ووفق السطر الأول من (is_bigger) بالبحث عن وجود حقيقة مباشرة في قاعدة المعطيات اسمها (bigger)، تنص على أن الفيل أكبر من القرد، فلا يجد. عندها ينتقل للسطر الثاني طارحاً على نفسه السؤال الآتي:

هل يوجد أي كائن حي آخر بحيث يحقق الحقيقة المباشرة التي تنص على أن الفيل أكبر منه، وأنه هذا الكائن الحي يحقق كونه أكبر من كائن آخر ، حتى الوصول إلى القرد بعلاقة متتابعة أي متعدية ؟.

بتتابع التنفيذ نجد أن الحصان يحقق أن الفيل أكبر منه، والحصان أكبر من حيوان آخر هو النمر يحقق كونه أكبر من القرد. عندها نحصل على الإجابة الصحيحة بنعم.

إن ما قام به البرولوغ جيد، ولكنه اقتصر على الإجابات المختصرة بكلمتي نعم ولا فقط. وهنا نقول:

ألا يمكن للبرولوغ أن يقوم بعمل أفضل وأكثر ذكاء ؟.

بالطبع هذا ممكن، فعلى سبيل المثال يمكننا معرفة أسماء جميع الحيوانات الموجودة في قاعدة المعطيات، والمحققة لشرط أن تكون أكبر من النمر وذلك بكتابة الاستعلام الآتي:

```
?- is_bigger(X, tiger).
X = horse.
```

يعبر (X) عن اسم أي متحول، وبالتالي فترجمة هذا الاستعلام؛ هي:

ما هي أسماء جميع الحيوانات الأكبر من النمر، والمعرفة طبعاً ضمن قاعدة المعطيات؟.

لا شك، أن الإجابة صحيحة ، فالحصان أكبر من النمر حسب الحقيقة التي عرفناها بشكل مباشر في قاعدة المعطيات. يمكن لنا الاكتفاء بهذه الإجابة وضغط زر الإدخال للانتقال إلى سطر جديد فحسب. ولكن المنطق الذهني يقول لنا إن الفيل أيضاً أكبر من النمر، ولكنه يحتاج إلى معالجة ذهنية منطقية ذكية نسبياً. وهذا بالضبط ما يمكن أن يقوم به البرولوغ حيث ما علينا سوى الضغط على زر الفاصلة المنقوطة لكي نحصل على إجابة صحيحة أخرى، وهكذا حتى تنتهي الإجابات الصحيحة، أو نضغط زر الإدخال، ليكون الوضع كما هو موضح:

```
?- is_bigger(X, tiger).
X = horse;
X = elephant;
false.
```

وكمثال آخر، لنستعلم عن الحيوانات الأصغر من النمر والأكبر من القرد، أي بتحقق الشرطين سوية، لذلك فعلينا استخدام الفاصلة؛ كما هو مبين:

```
?- is_bigger(tiger, X), is_bigger(X, monkey).
false.
```

طبعاً، الإجابة صحيحة حيث لا يوجد أي حيوان يحقق هذين الشرطين سوية ضمن قاعدة المعطيات.

ولكن ماذا لو كتبنا ما يأتي:

```
?- is_bigger(tiger, X); is_bigger(X, monkey).
```

X = sheep ;

X = monkey ;

X = tiger ;

X = elephant ;

X = horse ;

false.

نلاحظ وجود العديد من الإجابات حيث تعني الفاصلة المنقوطة ضمن الاستعلام علاقة (أو)، ومن الممكن أن يكون المتحول (X) في النصف الثاني من العلاقة متحولاً آخر مثل (Y) لنحصل عندها على ما يأتي:

?- is_bigger(tiger, X); is_bigger(Y, monkey).

X = sheep ;

X = monkey ;

Y = tiger ;

Y = elephant ;

Y = horse ;

false.

7-9- بعض المعلنات المعرفة مسبقاً في البرولوغ (Built-in Predicates):

يتوافر في البرولوغ، كما في أي لغة برمجية أخرى، مجموعة من أسماء المعلنات المعرفة مسبقاً (Built-in Predicates) في اللغة. لكن الملاحظ أن عدد هذه المعلنات قليل جداً، مقارنة مع لغات البرمجة التقليدية. عموماً، فإنها لا

تستخدم ضمن البرنامج وكأنها معلن قمنا بتعريفه ضمن قاعدة المعطيات، بل لها أصول في استخدامها، ولنتعرف على بعضها في هذه الفقرة.

1. المساواة (Equality):

إن محرف المساواة (=) له استخدامات عديدة، وربما كان أهم وأكثر المعلنات المعرفة مسبقاً استخداماً في البرولوغ. فعند كتابة العبارة الآتية في نمط الاستعلام:

```
?- moon = moon.
```

فإن البرولوغ يجيب بكلمة (true)، دالاً بذلك على نجاح اختبار التطابق بين طرفي إشارة المساواة. في حال عدم تطابق أحرف الطرف الموجود على يسار المساواة، مع الطرف الموجود على يمينها، فإن البرولوغ يجيب بكلمة (false).

أما عند كتابة العبارة:

```
?- X = moon.
```

فإنه يسند كلمة (moon) للمتحول (X).

ويمكن تنفيذ ما سبق بأحد الأشكال الآتية أيضاً:

```
?- =(moon, moon).
?- =(X, moon).
```

أما عند كتابة العبارة الآتية:

```
?- (X, moon, Sun) = (star, moon, Y).
X = star,
Sun = Y.
```

يقوم البرولوغ بعملية مطابقة لطرفي إشارة المساواة، مع إسناد المتحولات لما يناسبها من (Atom)، أو حتى المتحولات فيما بينها.

أيضاً من المناسب إيراد المثال الآتي:

```
?- (X, moon, X) = (star, moon, Y).
X = Y, Y = star.
```

2. ترجمة الملف (Compilation):

لتنفيذ البرنامج المكتوب بلغة البرولوغ يجب ترجمته أولاً، إما من القوائم الموجودة في البيئة البرمجية، أو باستخدام معلن معرف مسبقاً هو (consult/1)، حيث يعني رقم الواحد أن هذا المعلن يقبل وسيطاً (Argument) واحداً فقط.

يبين المثال كيفية ترجمة ملف محدد بإعطاء المسار كاملاً، مع الانتباه إلى استخدام إشارة التقسيم (/ slash)، وليس استخدام (\ backslash)؛ كما هو شائع في مسارات الملفات:

```
?- consult('D:/test1.pl').
% D:/test1.pl compiled 0.00 sec, 9 clauses
true.
```

بما أن إجابة البرولوغ (true)، فهذا يعني أن البرنامج قد ترجم بشكل صحيح ولا يوجد فيه أخطاء، بينما تتغير الرسالة عند وجود أخطاء.

3. الإظهار أو الإخراج (Output):

يحتاج المبرمج لتأمين سهولة التخاطب بين المستثمر والبرنامج أن يكتب بعض العبارات التوجيهية، أو يظهر بعض قيم المتحولات، أو أن يطلب إدخال قيم أخرى. يوجد عدة معلنات لتنفيذ هذا الأمر منها (write/1)، والتي تعيد كتابة ما

بين قوسين سواء كان (Term) أم متحولاً. يبين المثال استخدام تعليمة الإخراج بالحالتين:

```
?- write('Welcome to PROLOG'), nl.
Welcome to PROLOG
true.
```

قام البرولوغ بكتابة ما ضمن معلن (write)، وانتقل سطراً جديداً لوجود معلن معرف مسبقاً هو (nl/0)، ولكنه في هذا المثال ليس ذا فائدة. أما المثال الثاني:

```
?- X=fmee, write(X), nl.
fmee
X = fmee.
```

قام البرولوغ بإسناد قيمة (fmee) للمتحول (X)، ومن ثم تنفيذ عملية الكتابة، والانتقال لسطر جديد.

4. فحص نوعية عبارات البرولوغ (Checking the type of a Prolog term):

يوجد بعض المعلنات المعرفة مسبقاً في لغة البرولوغ، والتي تستخدم لمعرفة نمط العبارات أو المعرفات، وهذا ما توضحه الأمثلة الآتية:

```
?- atom(elephant).
true.
?- atom(Elephant).
false.
?- number(123).
```

```
true.
?- compound(X=smaller(cat)).
true.
```

مع الانتباه إلى عدم وجود معلن معرف مسبقاً باسم (variable)، بل باسم (var).

5. المساعدة في البرولوغ (Help):

تتوافر تعليمات المساعدة في جميع لغات البرمجة، وأيضاً فإن البرولوغ يتضمن معلناً معرفاً مسبقاً يستخدم لإعطاء بعض المساعدة عن المعلنات المعرفة مسبقاً في اللغة، كما هو مبين:

```
?- help(write).
true.
```

حيث تفتح نافذة شرح للمعلن (write).

كما يمكن عرض آخر (25) استعلام بكتابة (.h-?)، في نافذة الاستعلام.

7-10- إجابة الاستعلام (Answering Queries):

بعد التعرف على كيفية تعريف المعلنات في لغة البرولوغ ، وكيفية الاستعلام من قاعدة المعطيات، فإننا سنتعرف على كيفية معالجة الاستعلامات في البرولوغ للوصول إلى الإجابة.

1. التكافؤ (Matching):

يقال عن معرفين اثنين أنهما متكافئان عندما يتطابقان بشكل مباشر، أو أنه يمكن أن يتطابقا عند استخدام المتحولات. كما يتطابق متحولان حران لأنه يمكن أن يسند لهما القيمة نفسها. ويستثنى من ذلك المتحول (anonymous

(_ variable). فحتى نحصل على إجابة صحيحة في المثال المبين، يجب أن تتطابق المحارف تماماً لطرفي المساواة:

```
?- likes(apple, fmee, 7) = likes(apple, fmee, 7).
true.
```

أما عند استخدام المتحولات:

```
?- likes(X, fmee, 7) = likes(apple, fmee, 7).
X = apple.
```

فهنا لتحقيق التطابق فقد أسند للمتحول القيمة الموافقة لمكانه في المعلن.

أما في المثال الآتي، فإن الإجابة تكون بالنفي، لوجود تعارض في الإسناد للمتحول:

```
?- likes(X, fmee, 7) = likes(apple, Y, X).
false.
```

فلا يمكن أن يكون للمتحول (X) قيمتان مختلفتان هما (apple) و(7).

أما باستخدام المتحول (_)، فإنه يأخذ أي قيمة، دون أن يؤثر على صحة العبارة:

```
?- likes(_, fmee, 7) = likes(apple, Y, _).
Y = fmee.
```

وكأمثلة سريعة أخرى:

```
?- f(a, g(X, Y)) = f(X, Z).
X = a,
Z = g(a, Y).
```

?- f(a, g(X, Y)) = f(X, Z), Z = g(W, h(X)).

X = W, W=a,

Y = h(a),

Z = g(a, h(a)).

أحياناً لا نضع كلمة (true, false) في نهاية الإجابة لعدم الأهمية، أي لا ننسخ ما يظهر على شاشة البرولوغ تماماً. كما يختلف الإخراج حسب البيئة البرمجية، ففي المثال الأخير يمكن أن يكون الإخراج بالشكل:

?- f(a, g(X, Y)) = f(X, Z), Z = g(W, h(X)).

X = a,

Y = h(a),

Z = g(a, h(a)),

W = a,

true.

2. خطوات التنفيذ (Goal Execution):

عند كتابة الاستعلام فإن البرولوغ يحاول برهنة هذا الاستعلام والإجابة عنه، وهكذا عملية تسمى (goal execution). تمر عملية البرهنة بعدة مراحل حيث تبدأ بأسطر البرنامج لإيجاد حقيقة مباشرة للإجابة عن الاستعلام، وعند الحاجة للدخول في معلن علاقة بعد وجود المتحول في رأس العلاقة، فإن المتحول يدخل لجسم العلاقة ليصبح الهدف الجزئي هو برهنة جسم العلاقة، حتى إذا وجد في جسم العلاقة عدة معلنات، فإنه يصبح لدى البرولوغ عدة أهداف جزئية. عند تحقق المنطق الصحيح للأهداف الجزئية (ربما تكون العلاقة و/أو) ضمن جسم

العلاقة، عندها يصبح رأس العلاقة صحيحاً. أحياناً يكون هناك عدة طرق للحل، لذلك يختار البرولوغ الطريق الأول، خاصة إن كان حقيقة مباشرة. عند عدم التحقق من برهنة الاستعلام باختيار الطريق الأول، فإن البرولوغ ينتقل للطريق الآخر، وهكذا حتى تنتهي جميع الخيارات، أو يصل للحل الأول، فيظهره وينتظر إذا ما طلب المستثمر خياراً آخر.

لنأخذ المثال الآتي بصيغته النصية اللغوية أولاً، ثم نترجمه إلى البرولوغ:

إن جميع الرجال بشر.

إن عُمر رجل.

نستنتج من ذلك أن عمر من البشر. ولكن كيف تحول هذه العبارات النصية إلى لغة البرولوغ ؟.

الجملة الأولى تعبر عن قاعدة، فهي غير محددة، بل عامة. أما الجملة الثانية فتعبر عن حقيقة ثابتة محددة، لا خلاف عليها. وبالتالي يمكن كتابة البرنامج الآتي:

```
mortal(X) :-
        man(X).
man(omar).
```

وهي تعني أن المتحول (X) يكون من البشر، فيما إذا كان هذا المتحول رجلاً. وبالتالي فجميع الرجال بشر، ولكن ليس بالضرورة أن يكون أي بشري هو رجل، فمجموعة الرجال مجموعة جزئية من مجموعة البشر. بشكل ذهني، عرفنا أن عمر من البشر. ويمكن الاستعلام عن ذلك بكتابة ما يأتي:

```
?- mortal(omar).
true.
```

لنوضح كيف قام البرولوغ بإيجاد هذه النتيجة خطوة خطوة:

1) يحدد الهدف الابتدائي بأنه مبرهنة ((mortal(omar).

2) مسح عبارات البرنامج. يحاول البرولوغ مكافئة هذا الهدف بأول حقيقة مباشرة ممكنة، أو رأس قاعدة في البرنامج. لا يجد حقيقة، بل يجد رأس علاقة وحيدة، وبالتالي فقد أصبح لديه طريق وحيد يمكن أن يسلكه، وهذا الطريق يبدأ بعد عملية المكافئة بين المعرفات بإسناد (X= omar).

3) ينتقل المتحول إلى جسم العلاقة محتفظاً بقيمته الجديدة، أي ((man(omar).

4) يصبح الآن الهدف الجزئي للبرولوغ هو برهنة ((man(omar).

5) يعيد البرولوغ برهنة هذا الهدف الجزئي من جديد، محاولاً مكافئة الهدف مع حقيقة مباشرة، أو رأس علاقة ما في البرنامج.

6) يجد البرولوغ حقيقة مباشرة مكافئة ومطابقة للهدف المحدد، ما يعني أنه برهن على صحة الهدف الجزئي، وبالتالي الهدف الأعلى، وهو الهدف الابتدائي.

7-11- نصائح في كتابة البرامج (A Matter of Style) :

يجب أن يتوافق البرنامج الجيد مع أسس البرمجة الحديثة كالكفاءة والمتانة وترابط الأجزاء وسهولة القراءة. يتم بناء البرنامج بشكل هرمي من الأعلى إلى الأسفل مستعيناً بالقيم الابتدائية وحدودها. من أكثر ميزات لغة البرولوغ، قدرتها على كتابة برامج قصيرة، بعدد أسطر قليلة، وبكثافة برمجية عالية، لحل مشاكل

معقدة وصعبة الحل. فوق ذلك، تكون هذه البرامج مفهومة ومدركة نسبياً من قبل المبرمج أو المستخدم. ولتحقيق الفهم الجيد للبرنامج، حتى من قبل المبرمج نفسه، لا بد من استخدام الأسلوب الصحيح في كتابة البرامج. فيما يأتي بعض النصائح في كتابة البرامج:

1. النصيحة الأول في هذا المجال هي كتابة التعليقات المناسبة في الأماكن الصحيحة ضمن البرنامج، وقد تعرفنا على كيفية كتابة التعليقات سابقاً. فحتى المبرمج نفسه، إذا عاد بعد فترة للنص البرمجي الذي كتبه، فإنه يجد صعوبة في فهم ما كتب، ولماذا كتب هذا، ولماذا عرف ذلك المتحول؟.
2. يجب أن تكون مقاطع البرنامج صغيرة وتؤدي وظيفة واحدة فقط، والبرامج الفرعية يجب أن تكون قصيرة من أجل تسهيل عملية دراستها وصيانتها.
3. يفضل فصل عبارات البرنامج بسطر فارغ أو سطرين، مع كتابة اسم الفقرة كتعليق في بدايتها وضمن سطر مستقل.
4. البرنامج الجيد هو البرنامج المقروء، الذي يستطيع الإنسان قراءته بسهولة، وهذا الأمر يستدعي وضع الفراغات المناسبة وتنظيم الأسطر والجمل.
5. اكتب معلناً واحداً في السطر واستخدم الفراغ (Indentation) في بداية بعض العبارات، خاصة في جسم القاعدة، ولو كانت العبارة قصيرة، كالمثال الآتي:

```
blond(X) :-
        father(Father, X),
        blond(Father),
        mother(Mother, X),
```

blond(Mother).

6. استخدم الفواصل العادية ضمن (A compound term).

born(maha, damas, '01/01/1993')

7. لا تكتب عبارات معقدة، بل حاول قدر المستطاع أن تزيد عدد العبارات مقابل تبسيطها.

8. استخدم كلمات ذات معنى تعبر عن عمل المسمى، ولو اضطررت لاستخدام جمل كاملة.

7-12- مجموعة تمارين (Exercises):

فيما يأتي بعض التمارين البسيطة، التي ينصح بحلها يدوياً أولاً، وبعد ذلك التأكد من الحل بوساطة البرولوغ، ومناقشة السبب عند عدم تطابق الإجابتين.

طبعاً، وضعت الإجابات مباشرة بعد السؤال.

1. حدد أي من الكلمات الآتية يمكن أن تكون (Atom):

f, loves(kamal,mary), Mary, _c1, 'Hello', this_is_it.

true, false, false, false, true, true.

2. حدد أي من الكلمات الآتية يمكن أن تكون (Variable):

a, A, Paul, 'Hello', a_123, _, _abc, x2.

false, true, true, false, false, true, true, false.

3. ماذا يعطي الاستعلام الآتي:

```
?- f(a, b) = f(X, Y).
```

```
X = a,
Y = b.
```

4. ماذا يعطي الاستعلام الآتي:

```
?- loves(mary, kamal) = loves(Kamal, Mary).
```

```
Kamal = mary,
Mary = kamal.
```

5. إذا كانت الحقيقة الآتية (a(B, B)) فقط في البرنامج، فماذا يعطي الاستعلام الآتي:

```
?- a(1, X), a(X, Y), a(Y, Z), a(Z, 100).
```

```
false.
```

6. ماذا يعطي الاستعلام الآتي:

```
?- myFun(1, 2) = X, X = myFun(Y, Y).
```

```
false.
```

7. ماذا يعطي الاستعلام الآتي:

```
?- myFun(1, 2) = X, X = myFun(Y, Z).
```

```
X = myFun(1, 2),
Y = 1,
Z = 2.
```

8. ماذا يعطي الاستعلام الآتي:

```
?- f(a, _, c, d) = f(a, X, Y, _).
```

```
Y = c.
```

9. ماذا يعطي الاستعلام الآتي:

```
?- write('One '), X = write('Two ').
```

```
One
X = write('Two ').
```

10. ماذا يعطي الاستعلام الآتي:

```
?- write('One '); X = write('Two ').
One
true ;
X = write('Two ').
```

7-13- التعامل مع القوائم (List Manipulation):

تعد القوائم من أكثر بنى المعطيات استخداماً في البرولوغ، وقليلاً ما يكتب برنامج عملي بالبرولوغ دون الاستعانة بالقوائم.

1. توصيف القوائم (Notation):

تعرف القائمة ضمن قوسين مربعين (Square Brackets) كعناصر مستقلة مفصولة بوساطة فاصلة عادية، كما هو مبين:

```
?- X = [elephant, horse, tiger, sheep].
?- Y = [saturday, sunday, monday, tuesday, wednesday, thursday, friday].
```

القائمة الأولى (X) مكونة من أربعة عناصر (Atoms)، والقائمة الثانية (Y) مكونة من سبعة عناصر (Atoms). يمكن لعناصر القائمة (Elements) أن تكون أي عبارة (Term) صحيحة في البرولوغ، من العبارات الأربع (Atom, Number, Variable, Compound Term). كما يمكن أن تحتوي القائمة على قوائم ضمنية.

تعرف القائمة الفارغة بقوسين فارغين أي ([])، وكمثال عن ذلك:

```
?- Z = [elephant, [], X, parent(X, fadi), [a, b, c], f(22)].
```

بتعريف الحقيقتين الآتيتين ضمن نافذة البرنامج، وترجمة الملف:

```
weekdays([sun, mon, tue, wed, thu]).
weekends([fri, sat]).
```

يمكن الاستعلام عن أيام الأسبوع، للحصول على الناتج كما يأتي:

```
?- weekdays(Days).
Days = [sun, mon, tue, wed, thu].
```

كما يمكن الاستعلام عن أيام العطلة، للحصول على الناتج كما يأتي:

```
?- weekends([X, Y]).
X = fri,
Y = sat.
?- weekends([X, Y, Z]).
false.
```

من الواضح أن المتحول (Days) قائمة من أيام الأسبوع، أما المتحولات (X, Y) فهي عناصر القائمة، فعند استخدام الأقواس ضمن المعلن في الاستعلام تم إسناد قيم العناصر للمتحولات.

يوجد أسلوب ثانٍ يسمى (Internal Representation) لتعريف القائمة باستخدام الأقواس العادية، بالإضافة إلى النقطة (. Dot)، وهو أسلوب قليل الاستخدام، حيث تعد القائمة عبارة مركبة (Compound Term)، مع مراعاة تعريف القائمة الفارغة، بالشكل المبين:

```
.(a, .(b, .(c, [])))
```

وهي مطابقة للقائمة: [a, b, c]

فيمكن كتابة:

```
?- .(a, .(b, .(c, []))) = [X, Y, Z].
X = a,
Y = b,
Z = c.
?- .(a, .(b, .(c, []))) = X.
X = [a, b, c].
```

2. رأس وذيل القائمة (Head and Tail):

يسمى العنصر الأول من القائمة برأس القائمة (Head)، بينما تسمى بقية العناصر بذيل القائمة (Tail). لا تملك القائمة الفارغة رأساً، بينما يسمى العنصر في القائمة التي تحوي عنصراً وحيداً برأس القائمة، ويكون ذيلها قائمة فارغة. يستخدم للدخول إلى القائمة والوصول إلى عناصرها محرفاً خاصاً هو الخط الشاقولي (Vertical Line or Bar |). عند وضع الخط الشاقولي قبل آخر معرف في القائمة مباشرة، فهذا يعني أن هذا المعرف عبارة عن قائمة جزئية، وبالتالي تكون القائمة الكلية هي مجموعة العناصر ما قبل الخط الشاقولي، إضافة للقائمة الجزئية تباعاً. عند وجود عنصر وحيد قبل الخط فهذا العنصر يكون رأس القائمة، ويكون ما بعد الخط ذيل القائمة. وهذا ما يوضحه المثال الآتي:

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1,
Tail = [2, 3, 4, 5].
```

بالطبع، فإن أسماء المتحولات (Head, Tail) اختيارية، ولكن يفضل استخدام تسميات ذات معنى. كما أن متحول الذيل يكون قائمة، حتى لو كان قائمة فارغة. بينما الرأس فيجوز أن يكون عنصراً أو قائمة، كما يوضح المثال الآتي:

```
?- [[a, b, c], d, e, f] = [X | Y].
X = [a, b, c],
Y = [d, e, f].
```

تسمح طريقة التوصيف هذه بالوصول إلى عناصر القائمة كاملة، فإذا كان المطلوب الوصول للعنصر الثاني فقط، فإن ذلك يتحقق باستخدام المتحول المجهول (_)، كما هو مبين:

```
?- [a, b, c, d, e, f] = [_, X | _].
X = b.
```

كما يمكن الدخول إلى عناصر القائمة الفرعية بالشكل المبين:

```
?- [[a, b, c], d, e, f] = [[X | Y] | Z].
X = a,
Y = [b, c],
Z = [d, e, f].
```

يمكن باستخدام مفهوم الرأس والذيل، وبالاستعانة ببعض المعلنات المعرفة مسبقاً كتابة العديد من المعلنات الهامة والذكية.

يمكن الوصول أيضاً إلى عناصر القائمة بالشكل الآتي:

لنعرف الحقيقتين الآتيتين ضمن نافذة البرنامج، مع ترجمة الملف:

```
weekdays([sun, mon, tue, wed, thu]).
weekends([fri, sat]).
```

لنبين نتائج الاستفسارات الآتية:

```
?- weekdays([A | L]).
A = sun,
L = [mon, tue, wed, thu].

?- weekdays([A, B, C | L]).
```

```
A = sun,
B = mon,
C = tue,
L = [wed, thu].

?- weekends([A, B | L]).
A = fri,
B = sat,
L = [].
```

3. بعض المعلنات الجاهزة للقوائم (Built-in Predicates for List Manipulation):

يتضمن البرولوغ بعض المعلنات المعرفة مسبقاً، أي الجاهزة للتعامل السريع مع القوائم. عند طلب المساعدة لأي معلن معرف مسبقاً في البرولوغ، فإنه يعطي شرحاً للمعلن، مع الصيغة العامة للمعلن مع وضع خط مائل وعدد بعده يدل على عدد وسطاء المعلن. فمثلاً يدل الرقم (3) عند شرح المعلن (append/3) بأن له ثلاثة وسطاء، وهكذا لبقية المعلنات.

من أهم المعلنات التي تتعامل مع القوائم ما سيتم توضيحه أدناه:

1) (append/3):

يستخدم في دمج القوائم بشكل متتابع، وهو من المعلنات الهامة المعرفة مسبقاً في البرولوغ والخاصة بالقوائم، حيث يكون الوسيط الأول والثاني هما القوائم

المراد دمجها، والوسيط الثالث هو ناتج دمج القائمتين، كما هو موضح في الأمثلة الآتية:

عند الرغبة بالاستفسار عن صحة العبارة، يكتب المعلن دون متحولات، مثل:

```
?- append([a, b, c], [d, e, f], [a, b, c, d, e, f]).
true.
```

بما أن الوسيط الثالث هو ناتج دمج القائمتين، فإن الجواب هو (true).

أم عند الرغبة بالاستفسار عن قيمة المتحول (X) الذي يحقق ناتج دمج القائمتين فيكون الوسيط الثالث متحولاً، مثل:

```
?- append([1, 2, 3], [d, e, f, g], X).
X = [1, 2, 3, d, e, f, g].
```

كما يجوز أن يكون أي من الوسطاء متحولاً، ففي المثال الآتي يكون الوسيط الثاني متحولاً (Sec_Arg). يقوم البرولوغ بإيجاد القيمة المناسبة للمتحول (Sec_Arg)، حتى تتحقق صحة العلاقة، أي حتى يصبح الوسيط الثالث محققاً لناتج دمج الوسيطين الأول والثاني.

```
?- append([a, b, c], Sec_Arg, [a, b, c, d, 1, 2]).
Sec_Arg = [d, 1, 2].
```

كما يجوز أن يكون أكثر من وسيط من الوسطاء متحولاً. فيما يأتي الوسيط الأول والثاني عبارة عن متحولات. يقوم البرولوغ بإيجاد القيمة المناسبة للمتحول حتى تتحقق صحة العلاقة، أي حتى يصبح الوسيط الثالث محققاً لناتج دمج الوسيطين الأول والثاني. يوجد عدة خيارات ممكنة، لذلك يقوم البرولوغ

بإظهار جميع تلك الخيارات بشكل متتابع وفق طلب المستخدم، حيث تفصل الفاصلة المنقوطة بين تلك الخيارات.

```
?- append(Fer_Arg, Sec_Arg, [a, b, c]).
Fer_Arg = [],
Sec_Arg = [a, b, c] ;
Fer_Arg = [a],
Sec_Arg = [b, c] ;
Fer_Arg = [a, b],
Sec_Arg = [c] ;
Fer_Arg = [a, b, c],
Sec_Arg = [] ;
false.
```

2) (length/2):

يستخدم لحساب طول القائمة، وهو يتكون من وسيطين. يكون الوسيط الثاني هو طول القائمة المدخلة للمعلن:

```
?- length([elephant, [], [1, 2, 3, 4]], Length).
Length = 3.
```

ويمكن كتابة الشكل المبين، حيث أن ما يظهر عبارة عن متحولات غير محددة. تختلف الأعداد الظاهرة بعد حرف (G) بشكل دائم حسب محددات خاصة بالبرولوغ كإعادة تشغيل البرنامج، أو ترجمة ملف مثلاً:

```
?- length(X, 3).
X = [_G857, _G860, _G863].
```

3) (member/2) :

من أهم المعلنات الجاهزة الخاصة بالقوائم، حيث يختبر انتماء عنصر لقائمة، وهو ذو وسيطين:

```
?- member(sheep, [horse, tiger, sheep, monkey]).
true.
```

عند كون الوسيط الأول متحولاً، نحصل على جميع الخيارات الممكنة للمتحول:

```
?- member(X, [horse, tiger, sheep, monkey]).
X = horse ;
X = tiger ;
X = sheep ;
X = monkey.
```

يستخدم أحياناً كحلقة تكرار، كما هو مبين:

```
?- member(X, [1, 2, 3, 4, 5]).
X = 1 ;
X = 2 ;
X = 3 ;
X = 4 ;
X = 5.
```

4) (last/2):

يجيب هذا المعلن بالإيجاب، عند كون الوسيط الثاني هو العنصر الأخير في القائمة المدخلة كوسيط أول:

```
?- last([sun, 123, fine], fine).
true.
```

عند كتابة الوسيط الثاني كمتحول، فإنه يسند له آخر عنصر من القائمة:

```
?- last([1, 2, -45, ab, list], Last).
Last = list.
```

5) (reverse/2):

يستخدم لعكس ترتيب القائمة المدخلة، وإسناد ذلك للوسيط الثاني:

```
?- reverse([1, 2, 3, 4, 5], X).
X = [5, 4, 3, 2, 1].
```

6) (select/3):

يستخدم للقيام بعملية نفي التقاطع، أي يزيل العنصر المعطى كوسيط أول، من القائمة المعطاة كوسيط ثان، ويضع ما تبقى من القائمة في الوسيط الثالث، دون حذف التكرار في العنصر:

```
?- select(bird, [mouse, bird, fish, zebra], X).
X = [mouse, fish, zebra].
```

وكمثال آخر، عندما يكون الوسيط الأول والثالث متحولات، نحصل على النتائج المبينة:

```
?- select(Y, [mouse, bird, fish, zebra], X).
Y = mouse,
X = [bird, fish, zebra] ;
```

Y = bird,

X = [mouse, fish, zebra] ;

Y = fish,

X = [mouse, bird, zebra] ;

Y = zebra,

X = [mouse, bird, fish] .

7) (max_list/2):

يستخدم لإيجاد العدد الأعظمي ضمن القائمة:

?- max_list([1, 2, -45], Max).

Max = 2.

8) (min_list/2):

يستخدم لإيجاد أصغر عدد ضمن القائمة:

?- min_list([1, 2, -45], Min).

Min = -45.

9) (delete/3):

يستخدم لحذف عنصر هو الوسيط الثاني، من ضمن القائمة المعطاة كوسيط أول، ووضع الناتج في الوسيط الثالث:

?- delete([a, b, c, d, c], c, X).

X = [a, b, d].

10) (sum_list/2):

يستخدم لإيجاد ناتج جمع عناصر القائمة المعطاة كوسيط أول، ووضع الناتج في الوسيط الثاني:

```
?- sum_list([44, 5, 7], Sum).
Sum = 56.
```

11) (subset/2):

يستخدم لاختبار كون جميع عناصر القائمة المعطاة كوسيط أول، موجودة في القائمة المعطاة كوسيط ثان:

```
?- subset([a, b], [c, g, a, c, b]).
true.
```

4. تمارين محلولة (Exercises):

فيما يأتي بعض التمارين البسيطة، التي ينصح بحلها يدوياً أولاً، وبعد ذلك التأكد من الحل بوساطة البرولوغ، ومناقشة السبب عند عدم تطابق الإجابتين. مع ملاحظة أن الإجابات وضعت مباشرة بعد السؤال.

1) عرف معلناً باسم (analyze_list/1)، يقوم بما يأتي:

يأخذ قائمة كمدخل له، ويقوم بطباعة رأس وذيل القائمة مسبوقاً بجملة توضيحية.

في حال تمرير قائمة فارغة، فإنه يطبع رسالة مناسبة معبرة عن ذلك، وفي حال تمرير مدخل غير القائمة فإنه يجيب بالنفي.

الحل:

نعرف ضمن قسم البرنامج معلنين بالاسم نفسه، حيث يعالج المعلن الأول حالة القائمة الفارغة، ويعالج المعلن الثاني حالة القائمة غير الفارغة، بينما يترك للبرولوغ معالجة حالة المدخل غير القائمة.

```
analyze_list([]):-
        write('This is an empty list').
analyze_list([X|Y]):-
        write('This is the head of your list: '),
        write(X),
        nl,
        write('This is the tail of your list: '),
        write(Y).
```

بالاستفسار عن المعلن ضمن قسم الاستفسار نحصل على الإجابات الآتية:

```
?- analyze_list([sun, moon, star, earth]).
This is the head of your list: sun
This is the tail of your list: [moon, star, earth]
true.

?- analyze_list([]).
This is an empty list
true.
```

```
?- analyze_list(sky).
false.
```

2) عرف معلناً باسم (my_member/2)، يقوم بما يأتي:

فحص انتماء عنصر لقائمة، بمعنى آخر مطابق في عمله للمعلن المعرف مسبقاً (member/2)، ولكن دون استخدام المعلن (member).

الحل:

باستخدام المعلن (select)، واختبار الوسيط الثالث منه، فيما إذا لم يكن قائمة فارغة نحصل على المطلوب.

```
my_member(X, Y):-
        select(X, Y, Z),
        Z \= [].
```

كما يمكن إيراد طريقة أخرى دون استخدام أي معلن مسبق التعريف كما يأتي:

```
my_member(X, [X|L]).
my_member(X, [Y|L]):-
        my_member(X, L).
```

3) ليكن البرنامج المبين، ما هي حالات إعطاء الإجابة بالإيجاب عند الاستعلام عن (whoami(X)).

```
whoami([]).
whoami([_, _ | Rest]):-
        whoami(Rest).
```

الحل:

عندما تكون (X) قائمة بعدد زوجي من العناصر، عندها تكون الإجابة بالإيجاب.

```
?- whoami([a, 1, 67, star, uni, prolog]).
true.
?- whoami([a, 1, 67, star, uni]).
false.
```

4) عرف معلناً باسم (my_first/2)، يقوم بما يأتي:

إيجاد العنصر الأول من القائمة المعطاة كوسيط أول.

الحل:

بعض النسخ يتوافر فيها معلناً جاهزاً باسم (first/2)، أما النسخة المعتمدة في الكتاب فهي لا تتضمن هذا المعلن، والذي يعرف ببساطة بالشكل:

```
my_first([X | L], X).
```

إذن، يمكن تعريفه كحقيقة.

5) عرف معلناً باسم (my_last/2)، يقوم بما يأتي:

إيجاد العنصر الأخير من القائمة المعطاة كوسيط أول، دون استخدام المعلن الجاهز (last/2).

الحل:

نعرف حقيقة وعلاقة بالشكل:

```
my_last([X], X).
my_last([X1|L], X2):-
        my_last(L, X2).
```

بإضافة الحقيقة الآتية أيضاً إلى قاعدة المعطيات:

```
chara([a, b, c, d, e]).
```

يمكن الاستفسار بالشكل الآتي:

```
?- chara(X), my_last(X, L).
X = [a, b, c, d, e],
L = e .
```

6) عرف معلناً باسم (my_delete/3)، يقوم بما يأتي:

حذف عنصر هو الوسيط الثاني، من ضمن القائمة المعطاة كوسيط أول، ووضع الناتج في الوسيط الثالث، دون استخدام المعلن الجاهز (delete/3).

الحل:

نعرف حقيقة وعلاقتين بالشكل:

```
my_delete(X, [], []).
my_delete(X, [X|L], M):-
        my_delete(X, L, M).
my_delete(X, [Y|L], [Y|M]):-
        not(X=Y),
        my_delete(X, L, M).
```

يمكن الاستفسار بالشكل الآتي:

```
?- my_delete(a, [a, b, c, 3, a, rr], X).
X = [b, c, 3, rr] .
```

7) عرف معلناً باسم (my_subset/2)، يقوم بما يأتي:

اختبار وجود عناصر القائمة المعطاة كوسيط أول في القائمة المعطاة كوسيط ثان، دون استخدام المعلن الجاهز (subset/2).

الحل:

نعرف حقيقة وعلاقة بالشكل:

```
my_subset([], L).
my_subset([X|L1], L2):-
        member(X, L2),
        my_subset(L1, L2).
```

يمكن الاستفسار بالشكل الآتي:

```
?- my_subset([su, fr], [sa, su, mo, tu, we, th, fr]).
true .
```

7-14- التعابير الرياضية (Arithmetic Expressions):

تنفذ العمليات الرياضية في البرولوغ بطريقة مختلفة، عما هو مألوف في لغات البرمجة التقليدية.

1. المعامل (is) (The is-Operator for Arithmetic Evaluation):

تم التعرف سابقاً على بعض المعاملات مثل (\ ,/ ,- ,* ,+)، وهذه المعاملات تستخدم مباشرة كأي (Atom)، كما تستخدم كاسم معلن ((3, 5)+)،

ويمكن استخدامها ضمن التعبير الإعلاني (5+3)، مع الانتباه أحياناً للزوم وضع فراغات، كما يبين المثال الآتي:

```
?- +(3, 5)=+(X, Y).
ERROR: Syntax error: Operator expected
ERROR: +(3, 5)
ERROR: ** here **
ERROR: =+(X, Y) .
?- +(3, 5)= +(X, Y).
X = 3,
Y = 5.
```

وبالتالي فإن هذه المحارف ليست رياضية، أي لا تعني ما اعتدنا عليه بأنها تدخل ضمن العلاقات الرياضية، فمثلاً إذا كتبنا ما يأتي:

```
?- 5 = 2+3.
false.
?- X = 2+3.
X = 2+3.
```

فإنه من الواضح، عدم التعرف على هذه العبارات كعلاقات رياضية، وهذا يذكرنا بأن أحد المجالات التي لا ينصح باستخدام البرولوغ فيها، هي المعادلات والعلاقات الرياضية المعقدة.

ولحل هذه المشكلة، فإنه يستخدم المعامل (is) المعرف مسبقاً، لتنفيذ العمليات الرياضية كما هو مبين:

```
?- 5 is 2+3.
true.
?- X is 2+3.
X = 5.
?- 2+3 is 5.
false.
```

يلاحظ من المثال الأخير، عدم السماح بوضع العلاقة على يسار المعامل (is)، حيث أنه ينفذ ما على يمينه، ويسنده لما على يساره، إن كان متحولاً. أما إن كان عدداً فيجيب بالإيجاب عند تطابق الطرفين. كما يجب الانتباه إلى توافق نمط الأعداد في طرفي المعامل (is)، كما يبين المثال الآتي:

```
?- 2 is 1.0+1.0.
false.
?- 2.0 is 1.0+1.0.
true.
```

أو يتم استخدام المعامل (=:=)، الذي يقارن نتيجة الطرفين، بالشكل المبين:

```
?- 2 =:= 1.0+1.0.
true.
```

لا يمكن معاملة التعابير الإعلانية كتوابع في البرولوغ، فالتعبير يتحقق أو يفشل ولا يأخذ قيمة. باعتبار أن (Y) و(Z) هما متحولات خرج لتعابير إعلانية فلا يمكن كتابة ما يأتي:

```
?- Ans is (f(X, Y) + g(X, Z)).
```

ولكن تكتب بالشكل الصحيح الآتي:

```
?- f(X, Y), g(X, Z), Ans is Y + Z.
```

حيث ينفذ كلا المعلنين (f, g)، ومن ثم تستخدم (is) لجمع المتحولات المحلية الناتجة عن المعلنين.

إن المعامل (is) غير عكوس إذ يجب أن يكون الطرف الأيمن من المعامل (is) مرتبطاً. بتعريف معلن إيجاد مربع عدد بالشكل:

```
sqr(X, Y):-
        Y is X*X.
```

فإن الاستفسار الآتي ممنوع، لأن القسم الأيمن من المعامل (is) متحولات غير مرتبطة:

يستخدم المعلن (var/1) للتخلص من هذه المحدودية نوعاً ما، حيث يتحقق إذا كان وسيطه غير مرتبط. يوضح المثال الآتي كيفية تعريف معلن إيجاد مجموع وسيطين، ووضع الناتج في الوسيط الثالث، مع مراعاة عدم عكوسية المعامل (is).

```
my_sum(X, Y, S):-
        not(var(X)),
        not(var(Y)),
        not(var(S)),
        Z is (X + Y),
        Z = S.
my_sum(X,Y, S):-
```

```
        not(var(X)),
        not(var(Y)),
        var(S),
        S is (X + Y).
my_sum(X,Y, S):-
        not(var(X)),
        var(Y),
        not(var(S)),
        Y is (S – X).
my_sum(X,Y, S):-
        var(X),
        not(var(Y)),
        not(var(S)),
        X is (S – Y).
```

لا يمكن معالجة المسألة إذا وجد متحولان غير مرتبطان لوجود عدد لا نهائي من الارتباطات.

2. بعض التوابع والعلاقات الرياضية المحجوزة (Predefined Arithmetic Functions and Relations):

يوجد نوعان من معاملات البرولوغ الرياضية وهي التوابع والعلاقات. سنوضح بعض هذه المعاملات، ونترك للقارئ إمكانية الاستزادة من المراجع الخاصة.

1) التوابع:

إن الجمع والضرب هي أمثلة بسيطة عن التوابع الرياضية المعرفة مسبقاً (المحجوزة) في البرولوغ. تكتب المعادلات الرياضية في البرولوغ بشكل مشابه لما لما هو متفق عليه في الوسط الرياضي، كما تبقى العمليات الأساسية من جمع وطرح وضرب وقسمة بالرموز ذاتها. تراعى الأسبقية في المعادلة الرياضية بحيث تكون الأقواس هي الأعلى أولوية ومن ثم الضرب والقسمة فالجمع والطرح. إذن، الاختلاف الوحيد هو استخدام المعامل (is) بدلاً من إشارة المساواة. المثال البسيط الآتي يوضح ذلك:

```
?- Y is (5 + (10 - 8 * 2) / 3).
Y = 3.
```

للرفع إلى القوة يستخدم معامل الضرب بشكل مكرر (**)، بحيث يوضع الأساس على يساره والأس على يمينه.

يوجد العديد من التوابع المشهورة كإيجاد العدد الأكبر من مدخلين (max/2)، وإيجاد العدد الأصغر من مدخلين (min/2)، وإيجاد القيمة المطلقة (abs/1)، والجذر التربيعي (sqrt/1)، والجيب (sin/1)، وباقي القسمة (modulo) (mod/2).

يستخدم المعامل الآتي (//) لإيجاد القسم الصحيح من ناتج القسمة، وللتقريب لأقرب عدد صحيح (round/1)، ولتحويل العدد الصحيح إلى حقيقي (float/1).

فيما يأتي بعض الأمثلة التوضيحية:

```
?- A is 3**4.
A = 81.
```

```
?- A is max(23, 56).
A = 56.
?- A is min(23, 56).
A = 23.
?- A is abs(-23).
A = 23.
?- A is sqrt(23).
A = 4.795831523312719.
?- A is sin(pi/6).
A = 0.49999999999999994.
?- A is mod(10, 3).
A = 1.
?- A is (10 // 3).
A = 3.
?- A is round(35 / 10).
A = 4.
?- A is float( 10).
A = 10.0.
?-  float( 10.1).
true.
?- X is (max(2,6) + 2**3 +(abs(-4))).
X = 18.
```

2) العلاقات المنطقية:

تستخدم العلاقات الرياضية لمقارنة تعبيرين رياضيين. فعند تنفيذ عملية المقارنة الآتية (X > Y)، نحصل على إجابة بالإيجاب، أي بنعم، عندما تكون (X) أكبر من (Y). وهنا لا نحتاج للمعامل (is). كما تتوافر علاقة الأصغر (<)، والأصغر أو يساوي (=<)، والأكبر أو يساوي (>=)، وعدم المساواة (\=, =\=)، والمساواة المنطقية (=:=). مع الانتباه إلى الفرق بين المساواة المنطقية (=:=) والمساواة من ناحية التطابق والتكافؤ (=). والأمثلة الآتية توضح ذلك:

```
?- 2 ** 3 =:= 3 + 5.
true.
?- 2 ** 3 = 3 + 5.
false.
?- 3 =\= 31.
true.
?- 3 \= 31.
true.
?- 3 >= 2.
true.
```

أيضاً، فإن العلاقة (=:=) تنفذ حتى عندما يكون طرفا العلاقة أحدهما صحيحاً والآخر حقيقياً، وليس مثل (is) التي يتوجب فيها توافق الأنماط على طرفيها. كما هو مبين:

```
?- 2.0  =:= 2.
true.
?- 2.0  is 2.
false.
```

3. تمارين محلولة (Exercises):

فيما يأتي بعض التمارين البسيطة، والتي ينصح بحلها يدوياً أولاً، وبعد ذلك التأكد من الحل بوساطة البرولوغ، ومناقشة السبب عند عدم تطابق الإجابتين. مع ملاحظة أن الإجابات وضعت مباشرة بعد السؤال.

1) عرف معلناً باسم (distance/3)، يقوم بما يأتي:

حساب المسافة بين نقطتين في المحاور الإحداثية الثنائية الديكارتية، وفق النموذج المبين:

```
?- distance((-2.5, 1), (3.5, -4), X).
X = 7.81025.
```

الحل:

نعرف معلن علاقة بالشكل:

```
distance((X1, Y1), (X2, Y2), Dis) :-
        X is (X1-X2),
        Y is (Y1-Y2),
        Dis is sqrt(X**2 + Y**2).
```

2) عرف معلناً باسم (triangle/2)، يقوم بما يأتي:

طباعة محرف معطى في المعلن على شاشة الحاسوب، مكوناً شكل مثلث قائم قاعدته للأعلى وطولها محدد في المعلن، وفق النموذج المبين:

```
?- triangle(5, *).
*****
****
***
**
*
```

الحل:

نبدأ بتعريف معلن حقيقة وعلاقة (write1/2)، لكتابة محرف معطى كوسيط ثان، عدداً من المرات المعطاة كوسيط أول، على السطر نفسه. ثم يستخدم هذا المعلن ضمن معلن (triangle/2)، بالشكل:

```
write1(0,Y).
write1(X,Y):-
        X > 0,
        write(Y),
        X1 is X-1,
        write1(X1,Y).
triangle(0,Y).
triangle(N,Y):-
        N > 0,
        write1(N,Y),
```

```
N1 is N-1,
nl,
triangle(N1,Y).
```

3) عرف معلناً باسم (square/3)، يقوم بما يأتي:

طباعة محرف معطى في المعلن على شاشة الحاسوب، مكوناً شكل مربع طوله محدد في المعلن، وفق النموذج المبين:

```
?- square(6, '& ', 0).
& & & & & &
& & & & & &
& & & & & &
& & & & & &
& & & & & &
& & & & & &
```

الحل:

نبدأ بتعريف معلن حقيقة وعلاقة (write1/3)، لكتابة محرف معطى كوسيط ثان، عدداً من المرات المعطاة كوسيط أول، على السطر نفسه. أما الوسيط الثالث فيستخدم كعداد عام بين المعلنين. ثم يستخدم هذا المعلن ضمن معلن (square/3)، بالشكل:

```
write1(0,_, _):-
        nl,
        !.
```

```
write1(Num, Char, H1):-
        write(Char),
        Num1 is Num-1,
        write1(Num1, Char, H1).
square(Num,_, Num):-
        nl,
        !.
square(Num, Char, H):-
        H1 is H+1,
        write1(Num, Char, H1),
        square(Num, Char, H1).
```

استخدمت إشارة التعجب في هذا التمرين للخروج من التنفيذ، وهذا يفيد في عدم تعليق الحاسوب بعد التنفيذ الأول وإعادة طلب الاستمرار بالتنفيذ بضغط زر الفاصلة المنقوطة. كما استخدم معلن بثلاثة وسطاء، حيث استخدم الوسيط الثالث كعداد مساعد. كما تم التوفير بأسماء المتحولات باستخدام المتحول (_).

4) عرف معلناً باسم (fibonacci/2)، يقوم بما يأتي:

طباعة عدد فيبوناتشي من الرتبة المعطاة في المعلن على شاشة الحاسوب، علماً أن:

$$F_0 = 1 \quad ; \quad F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \ \ for\ n \geq 2$$

```
?- fibonacci(7, X).
X = 21
```

الحل:

```
fibonacci(0, 1):-
        !.
fibonacci(1, 1):-
        !.
fibonacci(Num, X):-
        Num1 is Num-1,
        Num2 is Num-2,
        fibonacci(Num1, X1),
        fibonacci(Num2, X2),
        X is X1+ X2.
```

5) عرف معلناً باسم (in_range /3)، يقوم بما يأتي:
فحص كون قيمة الوسيط الأول بين الوسيطين الثاني والثالث.

الحل:

```
in_range(X, Y, Z):-
        not(var(X)),
        not(var(Y)),
        not(var(Z)),
        X >= Y,
        X =< Z.
in_range(X, Y, Z):-
```

```
        var(X),
        not(var(Y)),
        not(var(Z)),
        Y =< Z,
        X is Y.
in_range(X, Y, Z):-
        Y =< Z,
        Y2 is Y + 1,
        in_range(X, Y2, Z).
```

عند طلب الاستفسار الآتي:

```
?- in_range(A, 2, 4).
A = 2 ;
A = 3 ;
A = 4 ;
false.
```

قام البرولوغ بتنفيذ القاعدة الثانية أولاً ووضع قيمة (A=2)، وبكتابة فاصلة منقوطة ينتقل للقاعدة الثالثة مستدعياً القاعدة الثانية وواضعاً قيمة (A=3)، وهكذا حتى يصل إلى (A=4).

7-15- التعقب الخلفي (Backtracking):

1. مفهوم التعقب الخلفي أو التراجعية:

في عمليات البحث عن الحل، أو التنقيب عن المعطيات، فإن البرنامج يصل إلى مفترق طرق، أي يصل إلى وجوب اختيار إحدى الطرق المتاحة من عدة احتمالات. باختيار أحد الطرق ربما يصل الطريق المختار ثانية إلى مفترق آخر، ليعود ويختار إحدى الاحتمالات، وهكذا حتى يصل إما إلى الحل المطلوب، أو إلى طريق مسدود. في الحالة الأولى، قد يكتفي المستخدم بالحل، وربما رغب بالبحث عن حل آخر، فبعض المسائل لها عدة حلول. وفي الحالة الثانية يتوجب العودة للوراء لاختيار طريق آخر ومتابعة البحث. إن خوارزمية العودة لطرق أخرى تسمى التراجعية أو التعقب الخلفي، وهي ذات أهمية وفائدة كبيرتين في حل العديد من المسائل الرياضية، وحتى الأحاجي المستخدمة للتسلية، كالكلمات المتقاطعة (Crosswords) والسودوكو (Sudoku). رغم صياغة مفهوم التعقب الخلفي منذ عام (1950) على يد الرياضي الأمريكي (D. H. Lehmer)، إلا أن أول لغة برمجية تضمنت معالجة العودية (Recursion) هي (SNOBOL) عام (1962)، حيث أن العودية هي استدعاء المعلن لنفسه، أي يرد اسم المعلن ضمن جسم المعلن نفسه.

ليكن المثال البسيط الآتي:

لنعرف معلناً (permutation/2) يقوم بإيجاد جميع التراتيب الممكنة لعناصر قائمة معطاة كوسيط أول، وإظهار أحد خيارات الترتيب في الوسيط الثاني:

```
permutation([], []).
permutation(List, [Element | Permutation]) :-
        select(Element, List, Rest),
        permutation(Rest, Permutation).
```

إن أبسط حالة هي كون القائمة فارغة، وتكون النتيجة قائمة فارغة أيضاً. وهذا ضروري كشرط لتوقف عملية البحث، فدائماً يجب البحث عن الحالة التي توقف عملية البحث، أو ما يسمى بشرط إنهاء العودية. الآن إذا كانت القائمة تحتوي على عناصر فإنه يتم الدخول للقاعدة وأول هدف جزئي (Subgoal) هو (select(Element, List, Rest)). والذي يعطى عادة (Atom) ولكنه هنا أعطي (Variable) ما يعني أن البرولوغ سوف يأخذ جميع الخيارات الممكنة، مبتدئاً بالعنصر الأول من القائمة، وواضعاً النتيجة في (Rest)، ليطلب ثاني هدف جزئي، وهو (permutation) ولكن بقائمة جديدة لا يوجد فيها العنصر الأول. تكون نتيجة التنفيذ بالشكل:

```
?- permutation([1, 2, 3], X).
X = [1, 2, 3] ;
X = [1, 3, 2] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
X = [3, 1, 2] ;
X = [3, 2, 1] ;
false.
```

ولزيادة التوضيح يمكن مراقبة خطوات التنفيذ باستخدام (trace/0)، كما يأتي:

```
?- trace, permutation([1, 2, 3], X).
   Call: (7) permutation([1, 2, 3], _G3897) ? creep
   Call: (8) lists:select(_G4012, [1, 2, 3], _G4024) ? creep
   Exit: (8) lists:select(1, [1, 2, 3], [2, 3]) ? creep
   Call: (8) permutation([2, 3], _G4013) ? creep
   Call: (9) lists:select(_G4015, [2, 3], _G4027) ? creep
   Exit: (9) lists:select(2, [2, 3], [3]) ? creep
   Call: (9) permutation([3], _G4016) ? creep
   Call: (10) lists:select(_G4018, [3], _G4030) ? creep
   Exit: (10) lists:select(3, [3], []) ? creep
   Call: (10) permutation([], _G4019) ? creep
   Exit: (10) permutation([], []) ? creep
   Exit: (9) permutation([3], [3]) ? creep
   Exit: (8) permutation([2, 3], [2, 3]) ? creep
   Exit: (7) permutation([1, 2, 3], [1, 2, 3]) ? creep
X = [1, 2, 3] ;
  Redo: (10) permutation([], _G4019) ? creep
   Call: (11) lists:select(_G4021, [], _G4033) ? creep
   Fail: (11) lists:select(_G4021, [], _G4033) ? creep
```

Fail: (10) permutation([], _G4019) ? creep

Redo: (10) lists:select(_G4018, [3], _G4030) ? creep

Fail: (10) lists:select(_G4018, [3], _G4030) ? creep

Fail: (9) permutation([3], _G4016) ? creep

Redo: (9) lists:select(_G4015, [2, 3], _G4027) ? creep

Exit: (9) lists:select(3, [2, 3], [2]) ? creep

Call: (9) permutation([2], _G4016) ? creep

Call: (10) lists:select(_G4021, [2], _G4033) ? creep

Exit: (10) lists:select(2, [2], []) ? creep

Call: (10) permutation([], _G4022) ? creep

Exit: (10) permutation([], []) ? creep

Exit: (9) permutation([2], [2]) ? creep

Exit: (8) permutation([2, 3], [3, 2]) ? creep

Exit: (7) permutation([1, 2, 3], [1, 3, 2]) ? creep

X = [1, 3, 2];

ويمكن المتابعة لآخر حل.

توقف (trace) باستخدام (nodebug). كما أنه من الأفضل اختيار الإظهار البياني لخطوات الحل إن توافر ذلك في البيئة البرمجية كما هو الحال في (SWI-Prolog).

2. تعريف القطع (Introducing Cuts):

يستخدم التعقب الخلفي بكثرة في برامج البرولوغ مضيفاً عملية القطع (Cuts) التي تتحكم بسير البرنامج، وتزيد من فعاليته باستخدام إشارة التعجب (!)، و(Conjunction) التي تربط بين المعلنات كعلاقة (و) المنطقية باستخدام (,)، و(Negation) التي تنفي، وتعكس باستخدام (+\)، و(Disjunction) التي تربط بين المعلنات كعلاقة (أو) المنطقية باستخدام (;).

تطرقنا في الفقرات السابقة إلى التراجعية (التعقب الخلفي)، دون الخوض في التفاصيل. حالياً سنتعرف أكثر على التراجعية مع حل بعض المشاكل باستخدام (Cuts).

هناك حالات ينفذ فيها التعقب الخلفي بشكل غير مستحب ويؤدي لإعطاء حلول إضافية غير صحيحة عند طلب المستخدم. بفرض تعريف المعلن (remove_duplicates/2) لحذف العناصر المتكررة في قائمة معطاة، كما هو مبين:

```
remove_duplicates([], []).
remove_duplicates([Head | Tail], Result) :-
        member(Head, Tail),
        remove_duplicates(Tail, Result).
remove_duplicates([Head | Tail], [Head | Result]) :-
        remove_duplicates(Tail, Result).
```

إن السطر الأول حقيقة تستخدم كشرط إنهاء العودية، وهو لحالة إدخال قائمة فارغة.

أما القاعدة الأولى فهي لاختبار وجود رأس القائمة ضمن ذيلها، أي تكرار الرأس (Head)، والنتيجة تكون باستدعاء المعلن عودياً لذيل القائمة دون الرأس لأنه مكرر.

أما القاعدة الثانية فهي لاستدعاء المعلن مع الاحتفاظ برأس القائمة.

تنفذ العملية بشكل صحيح، للحصول على الحل الأول. ولكن إذا قام المستخدم بالضغط على زر الفاصلة المنقوطة لإظهار حلول أخرى فإن خطأ ما سيظهر. سيقوم البرولوغ بتفرعات لوجود قاعدتين، وشجرة التفرعات الأولى ناتجة من القاعدة الأولى، حيث كلما وجد رأس في بقية القائمة فإنه يطرح (Discarded)، وعند التعقب الخلفي فإن جميع التفرعات سوف يتم زيارتها. وحتى عند تنفيذ القاعدة الأولى فإن الثانية يمكن أن تستدعى.

نتيجة هذه التفرعات تظهر النتائج الآتية:

```
?- remove_duplicates([a, b, b, c, a], List).
List = [b, c, a] ;
List = [b, b, c, a] ;
List = [a, b, c, a] ;
List = [a, b, b, c, a].
```

هذه النتائج الخاطئة، تبين أنه على البرولوغ عدم السماح للمستخدم بطلب حلول أخرى بعد الحل الأول.

يقدم البرولوغ حلاً لمسائل الفرز المشابهة لما سبق، فمن المناسب مقاطعة (Cut out) تفرعات التراجعية، ما يسمح بالحفاظ على الحل الصحيح فقط. يستخدم المحرف (!) لتنفيذ ذلك، ويعد كمعلن معرف مسبقاً، يوضع في أي مكان

ضمن جسم قاعدة، فيصبح هدف جزئي من سلسلة الأهداف الجزئية الموجودة في جسم القاعدة. إن المعلن (!) صحيح بشكل دائم، ولا يتم العودة إلى سلسلة الأهداف الجزئية الموجودة ما قبل (!) خلال التراجعية أبداً. بالعودة إلى المثال السابق وإضافة (!) إلى البرنامج ليصبح بالشكل المبين:

```
remove_duplicates([], []).
remove_duplicates([Head | Tail], Result) :-
        member(Head, Tail),
        !,
        remove_duplicates(Tail, Result).
remove_duplicates([Head | Tail], [Head | Result]) :-
        remove_duplicates(Tail, Result).
```

وهكذا فإنه كلما كان رأس القائمة عنصراً من ذيلها، فإن أول هدف جزئي في أول قاعدة أي (member(Head, Tail))، ستكون صحيحة (أي منطقياً true)، وبالتالي (!) صحيحة. وما عدا ذلك، فإنه يمكن التراجع للوصول لحل آخر. ولكن بمجرد تنفيذ (!) فلا يمكن العودة لما سبق. وبالتالي إذا طلب المستخدم حلاً آخر فلن يقبل ذلك:

```
?- remove_duplicates([a, b, b, c, a], List).
List = [b, c, a].
```

وبالتالي فإن تعريف (Cut) هي أنه كلما وردت (Cut) في جسم القاعدة فإن جميع الخيارات من لحظة تنفيذ رأس القاعدة وحتى لحظة ورود (Cut) سوف تلغى.

ليكن المثال الآتي:

```
beautiful(rama).
beautiful(maha).
beautiful(samira).
intelligent(samer).
intelligent(hasen).
fine_couple (Girl,Younker) :-
        beautiful(Girl),
        intelligent(Younker).
```

بطرح الاستفسار الآتي:

```
?- fine_couple(X,Y).
X = rama,
Y = samer ;
X = rama,
Y = hasen ;
X = maha,
Y = samer ;
X = maha,
Y = hasen ;
X = samira,
Y = samer ;
X = samira,
```

```
Y = hasen.
```

من الملاحظ إظهار جميع الخيارات الممكنة، ولكن بوضع (!) ضمن القاعدة بالشكل:

```
fine_couple (Girl,Younker) :-
        beautiful(Girl),
        !,
        intelligent(Younker).
```

فإن ناتج التنفيذ يصبح بالشكل:

```
?- fine_couple(X,Y).
X = rama,
Y = samer ;
X = rama,
Y = hasen.
```

لاحظ كيف أنه لم تنفذ التفرعات الممكنة ما قبل (Cut).

3. مشاكل القطع (Problems with Cuts):

صحيح أن (Cut) مفيدة جداً لتحديد سير برنامج البرولوغ، إلا أنها قد تسبب بعض المشاكل. لنبين ذلك بتعريف معلن (add/3) يقوم بإضافة عنصر معطى كوسيط أول، إلى قائمة معطاة كوسيط ثان، ووضع النتيجة في الوسيط الثالث، كما هو مبين:

```
add(Element, List, List) :-
        member(Element, List),
        !.
add(Element, List, [Element | List]).
```

وبالاستعلام نجد:

```
?- add(elephant, [dog, cat, rabbit], List).
List = [elephant, dog, cat, rabbit].
?- add(cat, [dog, cat, rabbit], List).
List = [dog, cat, rabbit].
```

النتائج جيدة وتعمل بالشكل المرغوب. فالمعلن (add) الثاني المكتوب كحقيقة لا ينفذ إذا ما نفذ الأول، أي كان العنصر موجوداً في القائمة، وبالتالي تبقى القائمة على ما هي عليه. وما عدا ذلك ينفذ الثاني. إن حذف (!) يؤدي إلى إضافة العنصر دائماً، أي اختبار المعلن الأول وإما الحفاظ على القائمة أو عدم تنفيذ شيء، وتنفيذ الثاني دائماً، بإضافة العنصر. وبالتالي إما أن نحصل على القائمة الأساسية ثم المضاف لها بشكل خاطئ، أو نحصل على القائمة المضافة بشكل صحيح. مع مراعاة تنفيذ المعلن (add) وفق ترتيب ورودهم في البرنامج.

ولكن ما هي المشكلة الموجودة، عندما نضع في الوسيط الثالث متحولاً، لا يوجد مشكلة. ولكن بإدخال قائمة ثابتة فإن البرولوغ يعمل بالشكل المبين:

```
?- add(a, [a, b, c, d], [a, a, b, c, d]).
true.
?- add(a, [a, b, c, d], [a, b, c, d]).
true.
```

من الواضح أن (!) منعت من تنفيذ العبارة الثانية. سنحل هذه المشكلة لاحقاً باستخدام (Negation). ولكن حالياً نحل المشكلة بتعديل بسيط على البرنامج، كما هو مبين:

```
add(Element, List, Result) :-
        member(Element, List),
        !,
        Result = List.
add(Element, List, [Element | List]).

?- add(a, [a, b, c, d], [a, a, b, c, d]).
false.
?- add(a, [a, b, c, d], [a, b, c, d]).
true.
```

المثال المبين لزيادة التوضيح:

```
add(Element, List, List) :-
        member(Element, List),
        !.
add(Element, List, [Element | List]).

?- add(WW, [dog, cat, rabbit], List).
WW = dog,
List = [dog, cat, rabbit] .
```

أما بحذف (!):

```
add(Element, List, List) :-
        member(Element, List).
add(Element, List, [Element | List]).

?- add(WW, [dog, cat, rabbit], List).
WW = dog,
List = [dog, cat, rabbit] ;
WW = cat,
List = [dog, cat, rabbit] ;
WW = rabbit,
List = [dog, cat, rabbit] ;
List = [WW, dog, cat, rabbit].
```

عموماً، يجب الحذر عند استخدام القطع (!)، ويجب استخدام عند الحاجة فقط، وبحيث لا يغير المعنى الوصفي للبرنامج، ويستخدم خاصة في جمل الشرط مثل:

```
if condition then
        sequence_of_statements_1
        else
        sequence_of_statements_2
```

تبرمج جملة الشرط السابقة في البرولوغ بالشكل:

```
head:-
        condition,
        !,
        sequence_of_statements_1;
        sequence_of_statements_2.
```

4. الرفض (Negation as Failure):

يتعامل البرولوغ عند الإجابة عن الاستفسارات غير الموجودة بالنفي، ولكن هذه الإجابة غير منطقية أحياناً. لتوضيح الرفض في البرولوغ نطرح المثال الآتي:

```
beautiful(rama).
beautiful(maha).
intelligent(samer).
intelligent(hasen).

?- beautiful(rama).
true.
?- intelligent(rama).
false.
```

أجاب البرولوغ بأن (rama) جميلة لوجود حقيقة تدل على ذلك. وأجاب بأنها غير ذكية لعدم وجود حقيقة تدل على أنها ذكية. ولكن كيف للمستخدم أن يعرف أن هذه الإجابة بالنفي ناتجة حقاً عن معالجة منطقية أو أنها بسبب عدم وجود حقيقة تدل عليها؟. هذا ما يسمى في البرولوغ بالرفض (Negation).

للحصول على إجابة بالإيجاب عن استعلام في البرولوغ، فإن البرولوغ يحاول إثبات ذلك الاستعلام من خلال الحقائق والقواعد التي لديه، وبالتالي فالإيجاب يعني الإجابة (true) لأن الاستعلام صحيح، وقد يعني إمكانية الإثبات. وهذا ينطبق على الإجابة السلبية (false)، حيث تكون الإجابة خاطئة أو لا يمكن إثباتها. إن هذا الأمر الذي لا يمكن إثباته يشار له بضبط الافتراضات (The closed word assumption)، وبالتالي كل ما لم يرد في البرنامج لا يفترضه. وهذا يدل على وجوب الحرص في كتابة البرنامج، فالإجابة السلبية قد تكون إيجابية في الحقيقة، ولكن عدم اعتبارها في إدخالات البرنامج أدى للإجابة السلبية.

5. النفي باستخدام (Fail):

تتميز البرولوغ باستخدام البحث التلقائي، وقد وفرت وسائل تمكن المستخدم من التدخل في طريقة سير التنفيذ وآلية البحث المعروفة. القطع إحدى الوسائل التي تعطي المستخدم القدرة على تغيير سير خطوات البرنامج، وذلك بمنع الاستمرار في البحث عن حلول أخرى عند توافر الحل المناسب. كما تحتوي البرولوغ على وسيلة أخرى تسمى النفي (fail) والتي تعد وسيلة تحكم بآلية البحث والتنفيذ، وهي تحمل المفهوم العكسي تماماً للقطع، حيث تسبب الاستمرار في البحث عن حلول أخرى. فعلى سبيل المثال:

```
student(ousama, 97).
student(ahmad, 86).
student(samer, 78).
student(kamal, 89).
```

```
student(rami, 87).
stu_mark:-
        student(S, M),
        write('name: '),
        write(S),
        write(' ----- '),
        write('mark: '),
        write(M),
        nl,
        fail.
stu_mark.
```

بكتابة الاستعلام الآتي:

```
?- stu_mark.
name: ousama ----- mark: 97
name: ahmad ----- mark: 86
name: samer ----- mark: 78
name: kamal ----- mark: 89
name: rami ----- mark: 87
true.
```

تم إظهار جميع الإجابات دون الحاجة للضغط على الفاصلة المنقوطة. كما أضيفت الحقيقية الأخيرة (stu_mark) كي يعطي في نهاية التنفيذ (true)، ولا يعطي (false).

6. المعامل (+\) (The \+ Operator):

من الهام أحياناً معرفة صحة هدف ما، ولكن أحياناً من الهام معرفة كونه خاطئاً. يستخدم المعامل (+\) في البرولوغ لتنفيذ نفي الهدف، بوضعه كسابقة للهدف المطلوب، والحصول بالنتيجة على إجابة هدف صحيحة، إذا كانت إجابة الهدف دون المعامل خاطئة، والعكس بالعكس. وبالتالي عند كون الإجابة خاطئة أو لا يمكن برهنتها عندها تصبح بوجود المعامل (+\) صحيحة. يقول القضاة " بريء حتى تثبت إدانته " (Innocent unless proven guilty)، وهذا ما يبينه المثال الآتي:

```
married(zaki, zakia).
married(sami, samia).
married(jamil, jamila).
single(Person) :-
        \+ married(Person, _),
        \+ married(_, Person).
?- single(zaki).
false.
?- single(rami).
true.
?- single(X).
false.
```

نشير أخيراً إلى أن المعامل (+\) يطبق لأي هدف (Goal) مقبول، والهدف قد يكون هدفاً جزئياً (Subgoal) في الاستعلام، أو في جسم القاعدة.

علماً أن الحقائق ورأس القاعدة ليست أهدافاً، وبالتالي لا يمكن رفض الحقيقة أو رأس القاعدة.

7-16- البرمجة العودية (Recursive Programming):

أشرنا سابقاً إلى العودية، وذلك من خلال التراجعية، ويجب معرفة أنها مصطلح غير مخصص للبرولوغ فقط، بل هي مصطلح هام في علوم الحاسوب والرياضيات. ولوجود بعض الصعوبة في هذا المصطلح، فإننا سنشرحه بشكل مبسط.

1. الاستقراء الكامل (Complete Induction):

يشبه مفهوم العودية مبدأ الاستقراء الرياضي. فللتعبير عن الأعداد الطبيعية نعتمد على المنطلق (Base Case) (n = 1)، لتصبح عبارة الأعداد الطبيعية تعتمد على (n+1). يمكن إعطاء مثال عن قانون حساب مجموع أول (n) عدد طبيعي بالقاعدة:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$n = 1: \quad \sum_{i=1}^{1} i = \frac{1(1+1)}{2} = 1 \ \ (base\ case)$$

$$n \rightsquigarrow n+1: \quad \sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$$

$$= \frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+2)}{2}$$

2. مبدأ العودية (The Recursion Principle):

إن الفكرة الأساسية في البرمجة العودية (Recursive Programming) هي أنه لحل مشكلة معقدة، نقوم بإيجاد حل أبسط حالة للمشكلة، ونضع قاعدة لتحويل المشكلة المعقدة إلى بسيطة.

وبتعبير آخر، نقدم حلاً لحالة (Base Case) مع قاعدة عودية أو خطوية. بعد ذلك نضع خوارزمية أو برنامج لحل المشكلة. إذن، فالاستقراء يبدأ من منطلق أو قاعدة أساسية ليعممها على بقية الحالات. أما العودية فهي بحساب تابع لأي حالة انطلاقاً من الأعلى نحو الأدنى أي الحالة القاعدية (Base Case).

3. مثال بسيط:

من أشهر الأمثلة المستخدمة في شرح العودية هو مثال حساب عاملي عدد طبيعي (Factorial n!)، والذي يعرف كمضاريب الأعداد من الواحد وحتى (n)، كما هو مبين:

$$1! = 1 \qquad (base\ case)$$
$$n! = (n-1)!.n \quad for\ n > 1 \qquad (recursion\ rule)$$

لحساب قيمة (5!)، فإننا تمرر إلى الجزء الثاني أربع مرات حتى تصل للحالة القاعدية، ومنها تحسب النتيجة الكلية، وهذه هي العودية.

تحل هذه المسألة في لغة الجافا بالشكل:

```
public int factorial(int n) {
if (n == 1) {
return 1;                    // base case
} else {
  return factorial(n-1) * n;       // recursion step
}
}
```

أما في لغة البرولوغ:

```
factorial(1, 1).              % base case
factorial(N, Result) :-       % recursion step
        N > 1,
        N1 is N - 1,
        factorial(N1, Result1),
        Result is Result1 * N.
```

4. مجموعة تمارين (Exercises):

فيما يأتي بعض التمارين البسيطة عن تعريف معلنات عودية.

1) عرف معلناً (len/2)لحساب طول سلسلة، بشكل مشابه للمعلن المعرف مسبقاً (length/2).

الحل:

يوجد عدة حلول سوف نناقشها تباعاً.

1. الطريقة الأولى:

```
len([], N) :-
        write(N),
        !.                  % base case
len([ _ | Tail], N) :-      % recursion step
        N1 is N + 1,
        len(Tail, N1).
```

وبالتالي يجب الانتباه دائماً إلى وجوب وضع شرط توقف العودية، أو ما سمي (Base Case). مع الحرص على اقتراب الحل من الشرط، خطوة بعد أخرى.

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- len([a, b, 1, 6, h], 0).
5
```

يفترض في هذا الحل أن يكون الوسيط الثاني (0) دائماً، وهذا يعني أنه لا يمكن إسناد النتيجة إلى متحول كما هو الحال في المعلن المعرف مسبقاً (length/2). كما أنه إذا قام المستخدم بوضع عدد غير الرقم (0)، فإن النتيجة سوف تكون أكبر من القيمة الحقيقية بمقدار قيمة العدد، كما هو مبين في الاستدعاء الآتي:

```
?- len([a, b, 1, 6, h], 12).
17
```

2. الطريقة الثانية:

```
len([], 0).
len([X], 1).
len([X, Y], 2).
len([X, Y, Z | Tail], N) :-
        len(Tail, N2),
        N is N2 + 3.
```

يمتاز هذا الحل بإمكانية أن يكون الوسيط الثاني عدداً أو متحولاً بما يشب تماماً المعلن المعرف مسبقاً (length/2). كما يمتاز بسرعة حساب طول القوائم القصيرة جداً، لاستخدامه ما يسمى في البرولوغ بالاستطراد في التعريف (Redundancy in Definitions)، حيث تم تعريف المعلن كحقائق عندما تكون قصيرة الطول. وبما أن سرعة التحقق من الحقيقة أعلى من سرعة تنفيذ العلاقة فإن سرعة حساب طول القوائم القصيرة جداً تكون مرتفعة.

2) عرف معلناً (sum1/2) لحساب عدد مراتب عدد، وطباعته على الشاشة.

الحل:

```
sum1(0, S) :-
        write(S),
        nl,
        !.
sum1(N, S) :-
```

```
        N1 is N div 10,
        S1 is S + 1,
        sum1(N1, S1).
```

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- sum1(14652, 0).
5.
```

إن عدم وضع إشارة التعجب (!)، يؤدي إلى إعطاء النتيجة (5) كأول حل، ثم (6) عندما يطلب المستخدم حلاً آخر، ثم (7)، وهكذا دون توقف. وبالتالي تكون الإجابة الأولى هي الصحيحة فقط.

3) عرف معلناً (sum2/2) لحساب مجموع مراتب عدد، وطباعته على الشاشة.

الحل:

```
sum2(0, S) :-
        write(S),
        nl,
        !.
sum2(N, S) :-
        N1 is N div 10,
        N2 is N mod 10,
        S1 is N2 + S,
        sum2(N1, S1).
```

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- sum2(148, 0).
13.
```

4) عرف معلناً (inverse/1) لعكس عدد يدخله المستخدم، وطباعته على الشاشة.

الحل:

```
inverse(0) :-
        nl,
        !.
inverse(N) :-
        N1 is N div 10,
        N2 is N mod 10,
        write(N2),
        inverse(N1).
```

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- inverse(6785, 0).
5876.
```

5) عرف معلناً (gcd/2) لإيجاد القاسم المشترك الأكبر لعددين، وطباعته على الشاشة.

الحل:

```
gcd(X, 0) :-
        write('g.c.d of tow numbers is: '),
        write(X),
        nl,
        !.
gcd(X, Y) :-
        Y1 is X mod Y,
        gcd(Y, Y1).
```

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- gcd(44, 55).
g.c.d of tow numbers is: 11
```

6) عرف معلناً (fun/2) لطباعة السلسلة المبينة على الشاشة:

(2, 8, 26, 80, 242, 728, …….).

الحل:

```
fun(0, _) :-
        !.
fun(N, K) :-
        N1 is N - 1,
        K1 is K * 3 + 2,
```

```
        write(K1),
        write(',  '),
        fun(N1, K1).
```

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- fun(8, 0).
2,  8,  26,  80,  242,  728,  2186,  6560.
```

يمكن كتابة البرنامج ببدايات مختلفة، مثل:

```
fun(0, _).
fun(N, K) :-
        N \== 0,
```

أو مثل:

```
fun(0, _).
fun(N, K) :-
        N =\= 0,
```

أو مثل:

```
fun(0, _) :-
        [].
fun(N, K) :-
        N \== 0,
```

ولا يوجد فرق، سوى أنه يعطي الناتج ثم (true)، ثم يزيد فحص حالة أخرى، ويكتب (false).

7) عرف معلناً (my_sort/2) لفرز الأعداد تصاعدياً، بشكل مشابه للمعلن المعرف مسبقاً (sort/2).

الحل:

```
my_sort([], []).
my_sort([X|L1], L2):-
        delete(L1, X, H),
        my_sort(H, L3),
        insert_item(X, L3, L2).
insert_item(X, [], [X]).
insert_item(X, [Y|L], [X, Y|L]):-
        X < Y.
insert_item(X, [Y|L1], [Y|L2]):-
        X >= Y,
        insert_item(X, L1, L2).
```

يمكن تجريب المعلن باستدعائه بالشكل:

```
?- my_sort([2, 1, 6, 4, 5, 7, 3, 2, 4, 4, 9, 8], Result).
Result = [1, 2, 3, 4, 5, 6, 7, 8, 9] .
```

تم تعريف المعلن (insert_item) ذي الوسطاء الثلاثة من أجل إدخال العنصر المعطى كوسيط أول في القائمة المعطاة كوسيط ثان ووضع الناتج في القائمة المعطاة كوسيط ثالث.

7-17- بعض المسائل الهامة في البرولوغ:

تم اختيار بعض المسائل التطبيقية العامة، وهي من نوع المشاكل التي يفضل حلها باستخدام لغات الذكاء الصنعي.

1. أحجية سارق الفندق (Hotel Stealer):

تنص الأحجية على أنه في إحدى الفنادق الراقية، قام أحد النزلاء بإبلاغ الجهات المختصة، عن عملية سرقة لغرفته. وبعد إجراء التحقيقات المستفيضة، ظهرت مجموعة من المعطيات والحقائق؛ هي:

1) إن لون شعر السارق بني.
2) تتزين الآنسة نور بخاتم ألماس جميل.
3) يرتدي السيد حسن ساعة فاخرة.
4) تنزل الآنسة نور في الغرفة رقم /12/.
5) ينزل السيد خالد في الغرفة رقم /10/.
6) ينزل السيد سامر في الغرفة رقم /16/.
7) ينزل الشخص في الغرفة رقم /14/، إذا كان لديه ساعة فاخرة.
8) إذا كان السيد خالد مرتدياً قفازات، فإن السيد سامر يضع نظارات.
9) إذا كان السيد سامر مرتدياً قفازات، فإن السيد خالد يضع نظارات.
10) يرتدي الشخص قفازات، إذا كان ينزل في الغرفة رقم /16/.
11) يكون لون شعر الشخص أسوداً، إذا كان ينزل في الغرفة رقم /14/.
12) يكون لون شعر الشخص أشقراً، إذا كان ينزل في الغرفة رقم /12/.

13) يكون لون شعر الشخص بنياً، إذا كان يضع نظارات.

14) يكون لون شعر الشخص أحمراً، إذا كان يرتدي قفازات.

والمطلوب:

1) كتابة برنامج بلغة البرولوغ لمعرفة السارق، اعتماداً على الحقائق السابقة.

2) تحديد الحقائق غير اللازمة لحل المسألة، ووضع إشارة (%) قبلها.

الحل:

بقراءة المسألة بتأنٍ يمكن استنتاج وجود أربعة نزلاء بأسماء مختلفة يشتبه بهم، وكل منهم ينزل في غرفة مستقلة، ويرتدي شيئاً مميزاً، وله لون شعر مختلف أيضاً. يبين الجدول الآتي تلك المعلومات على شكل مجموعات مستقلة:

أسماء النزلاء	(نور، حسن، خالد، سامر)
أرقام الغرف	(10, 12, 14, 16)
الشيء المميز	(ألماس، ساعة، قفازات، نظارات)
لون الشعر	(بني، أسود، أشقر، أحمر)

من الواضح وجود مجموعة من الحقائق ومجموعة من القواعد. يفضل تسمية الحقائق والقواعد بشكل معبر فمثلاً المعلن (name_room/2) للربط بين اسم الشخص ورقم غرفته، وله وسيطان؛ هما الاسم، ورقم الغرفة.

بناء على ذلك تكتب الحقيقة السادسة (ينزل السيد سامر في الغرفة رقم /16/) بالشكل:

```
name_room(samer,16).
```

أما العلاقات فتكتب اعتماداً على تعريف الحقائق. فعلاقة (ينزل الشخص في الغرفة رقم /14/، إذا كان لديه ساعة فاخرة) تكتب بالشكل:

```
name_room (X,14):-
        name_w(X, watch).
```

ليصبح البرنامج كاملاً وبنفس ترتيب ورود الحقائق والقواعد بالشكل:

```
name_room(samer, 16).
name_wear(nour, ring).%
name_wear(hasen, watch).%
name_room(nour, 12).%
name_room(khaled, 10).%
name_room(samer, 16).
name_room(X, 14):-
        name_wear(X, watch).%
name_wear(samer, eyeglass):-
        name_wear(khaled, gloves).%
name_wear(khaled, eyeglass):-
        name_wear(samer, gloves).
name_wear(X, gloves):-
        name_room(X, 16).
name_hair(X, black):-
        name_room(X, 14).
name_hair(X, blond):-
```

```
        name_room(X, 12).
name_hair(X, brown):-
        name_wear(X, eyeglass).
name_hair(X, red):-
        name_wear(X, gloves).
```

وتكون معرفة السارق بكتابة الاستفسار الآتي:

```
?- name_hair(X, brown).
X = khaled.
```

كان بالإمكان إضافة القاعدة الآتية للبرنامج:

```
steal(X):-
        name_hair(X, brown).
```

والاستعلام عن السارق بكتابة الاستفسار الآتي:

```
?- steal(X).
X = khaled.
```

ويجب ملاحظة ما يأتي:

يعطي مترجم البرولوغ رسالة تنبيه وليست رسالة خطأ، عند ترجمة البرنامج السابق، وذلك لكتابة أسماء الحقائق والقواعد المتشابهة باسم المعلن بشكل متتابع، ويمكن التجاوز عن الرسالة دون أن تسبب أي مشكلة. وهنا قمنا بعدم مراعاة هذه النقطة لكتابة البرنامج بنفس ترتيب ورود المعطيات في نص المسألة النثري.

كما أن بعض المعطيات غير هامة لحل المسألة ويمكن وضع إشارة (%) قبلها لتصبح تعليقاً. وقد حددت هذه المعطيات بكتابة إشارة (%) بعد العبارة المعبرة عنها.

يجب الانتباه إلى النص النثري، فسواء بدأت الجملة بأداة الشرط (إذا)، أم أخرت أداة الشرط للجزء الثاني من الجملة، فإن العلاقة المقابلة في البرولوغ لا تتأثر. فمثلاً عند القول:

إذا كان السيد سامر مرتدياً قفازات، فإن السيد خالد يضع نظارات.

كتبت بالشكل:

```
name_wear(khaled, eyeglass):-
        name_wear(samer, gloves).
```

أما لو كتبت بالشكل:

```
name_wear(samer, gloves):-
        name_wear(khaled, eyeglass).
```

فإن المسألة لن تحل. وهذا مشابه لحالة كتابة:

```
car(X):-
        vehicle(X).
```

فهذا يعني أن كل مركبة هي سيارة. ولكن الصحيح هو أن كل سيارة هي مركبة وبالتالي يجب كتابة ما يأتي:

```
vehicle(X):-
        car(X).
```

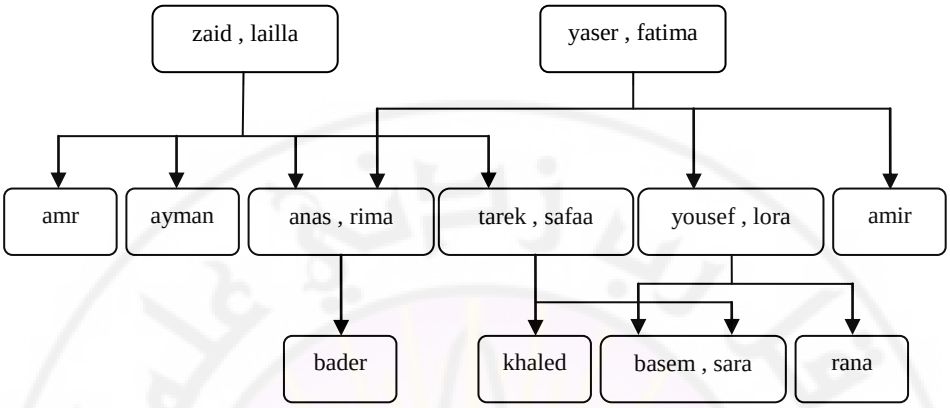
في النهاية يكون البرنامج بعد مراعاة الترتيب الذي لا يظهر رسالة التنبيه، وإهمال المعطيات غير الهامة بالشكل:

```
name_wear(khaled, eyeglass):-
    name_wear(samer, gloves).
name_wear(X, gloves):-
    name_room(X, 16).

name_room(samer, 16).
name_hair(X, black):-
    name_room(X, 14).
name_hair(X, blond):-
    name_room(X, 12).
name_hair(X, brown):-
    name_wear(X, eyeglass).
name_hair(X, red):-
    name_wear(X, gloves).
```

2. شجرة العائلة (Family Tree):

من المسائل كثيرة الورود في الذكاء الصنعي لما تقدمه من استفسارات صعبة الحل عن الأقارب والفروع والأصول، مع إمكانية إدخال علم التوريث وقواعده الصعبة للحصول على برنامج سهل الاستخدام، يحل العديد من المشاكل. لتكن شجرة العائلة المبينة بشكل تخطيطي في الشكل(7-5)، مع ملاحظة أن السهم يتجه من الآباء نحو الأبناء، ويقع رأس السهم فوق الابن مباشرة.



الشكل(7-5) مخطط شجرة العائلة

والمطلوب:

1) اكتب برنامجاً بلغة البرولوغ لتوصيف المسألة بأقل عدد ممكن من الحقائق.

2) عرف المعلنات الآتية:

1) (man(X)): لتحديد جنس الرجال.

2) (is_the_son_of(X,Y)): لتحديد الأبناء الذكور.

3) (is_the_daughter_of(X,Y)): لتحديد الأبناء الإناث.

4) (is_the_mother_in_law(X,Y)): لتحديد الحماة (أم الزوج أو الزوجة).

5) ((is_the_father_in_law(X,Y)): لتحديد الحمو (والد الزوج أو الزوجة).

6) ((is_grandson(X,Y)): لتحديد الأحفاد الذكور.

7) ((is_granddaut(X,Y)): لتحديد الأحفاد الإناث.

8) ((small_child(X)): لتحديد كل الأحفاد من الجنسين ، ولكن من الأبناء الذكور.

9) ((is_uncle_of(X,Y)): لتحديد العم.

الحل:

يجب مراعاة عدم تكرار المعلومات بزيادة الحقائق خلال الحل، مع عدم الإقلال منها. وبالتالي يجب كتابة مجموعة من الحقائق، التي يمكن بقراءتها فقط، إعادة رسم مخطط شجرة العائلة كما هو وارد، وباستخدام جميع الحقائق ودون زيادة أو نقصان. فمثلاً يمكن وصف الذكور كحقائق ووصف الإناث كقاعدة. كما يتم وصف العلاقة الزوجية كحقائق، ووصف علاقة الأبوة كحقائق أيضاً. أما الأمهات فتوصف كقاعدة.

قبل إيراد البرنامج كاملاً يجب شرح المعلنات المستخدمة.

يبدأ البرنامج في المرحلة الأولى بتعريف الحقائق. الحقيقة الأولى هي معلن نوع لتحديد جميع الذكور (male/1). بعد ذلك يعرف معلن (is_the_father_of/2) لتحديد الآباء والأبناء، حيث يكون الوسيط الأول هو الأب والثاني هو الابن، سواء الذكر أم الأنثى. بعد ذلك يعرف معلن

(is_the_husband_of/2) لتحديد الأزواج، حيث يكون الوسيط الأول هو الزوج والثاني هو الزوجة. وبذلك تكون قد اكتملت الحقائق كاملة، ويمكن من خلال هذه الحقائق فقط، إعادة رسم شجرة العائلة دون زيادة أي حقيقة.

في المرحلة الثانية يتم تعريف معلنات علاقة غير مطلوبة بشكل مباشر، ولكنها لازمة لتعريف بقية المعلنات. تكون البداية من المعلن (is_the_mother_of/2) لتحديد الأمهات والأبناء، حيث يكون الوسيط الأول هو الأم والثاني هو الابن، سواء الذكر أو الأنثى. من الواضح عدم وضع معلن الأم كحقيقة مثل معلن الأب لأنه يمكن تعريفه كعلاقة وهذا أكثر اختصاراً للبرنامج. ثم يعرف المعلن (female/1) كعلاقة لتحديد جميع الإناث. من الملاحظ تعقيد هذه العلاقة لتأخذ جميع الحالات، فقد كان من الممكن تعريفها باستخدام (not(man(X))) فقط، لكن في هذه الحالة سوف يعد كل اسم لم يعرف على أنه ذكر، بأنه أنثى، ولو لم يكن وارداً في قاعدة المعطيات. أما بالشكل الحالي فالأنثى هي الشخص الذي له أب وليس ذكراً، أو هو الوسيط الثاني في علاقة الأزواج وليس له أب في قاعدة البيانات. ثم يعرف المعلن (parent/2) كعلاقة لتحديد الوالد سواء الأب أم الأم، حيث يكون الوسيط الأول هو الوالد والثاني هو الابن، سواء الذكر أو الأنثى . ثم يعرف المعلن (is_the_brother_of/2) كعلاقة لتحديد الأخوة الذكور لاستخدامه في علاقة العم، حيث يكون الوسيط الأول هو أحد الأخوة والثاني هو أخو الوسيط الأول. من الملاحظ في معلن الأخوة اختبار عدم تساوي (X, Y)، كي لا يعطي الشخص ونفسه كأخوة. كما أنه يتم اختبار الأخوة من الأب وليس الأشقاء حصراً.

في المرحلة الثالثة يتم تعريف المعلنات المطلوبة بشكل مباشر في المسألة. تكون البداية من المعلن (man/1) لتحديد لتحديد جنس الرجال ، وهو

مطابق للمعلن الحقيقة (male/1). ثم يعرف المعلن (is_the_son_of/2) كعلاقة لتحديد الأبناء الذكور ، حيث يكون الوسيط الأول هو الابن الذكر والثاني هو الوالد، سواء الأب أم الأ م. ثم يعرف المعلن (is_the_daughter_of/2) كعلاقة لتحديد الأبناء الإناث ، حيث يكون الوسيط الأول هو الابنة والثاني هو الوالد، سواء الأب أم الأ م. ثم يعرف المعلن (is_the_mother_in_law/2) كعلاقة لتحديد الحماة ، حيث يكون الوسيط الأول هو الحماة والثاني هو زوجة الابن أو زوج الابنة. ثم يعرف المعلن (is_the_father_in_law/2) كعلاقة لتحديد الحم و، حيث يكون الوسيط الأول هو الحمو والثاني هو زوجة الابن أو زوج الابنة. ثم يعرف المعلن (is_grandson/2) كعلاقة لتحديد الأحفاد الذكور ، حيث يكون الوسيط الأول هو الأحفاد والثاني هو الجد أو الجدة. وتكون الأحفاد سواء من جهة الأب أم الأم. ثم يعرف المعلن (is_granddaut/2) كعلاقة لتحديد الأحفاد الإناث، حيث يكون الوسيط الأول هو الأحفاد والثاني هو الجد أو الجدة. وتكون الأحفاد سواء من جهة الأب أم الأم. ثم يعرف المعلن (small_child/1) كعلاقة لتحديد كل الأحفاد من الجنسين ولكن من الآباء الذكور، حيث يكون الوسيط الوحيد هو الأحفاد . ثم يعرف المعلن (is_uncle_of/2) كعلاقة لتحديد العم ، حيث يكون الوسيط الأول هو العم والثاني هو ابن الأخ أو ابنة الأخ.

وبذلك يكون البرنامج بالشكل الآتي:

```
male(zaid).
male(yaser).
male(amr).
male(ayman).
```

male(anas).
male(tarek).
male(yousef).
male(amir).
male(bader).
male(khaled).
male(basem).

is_the_father_of(zaid, amr).
is_the_father_of(zaid, ayman).
is_the_father_of(zaid, anas).
is_the_father_of(zaid, tarek).

is_the_father_of(yaser, rima).
is_the_father_of(yaser, yousef).
is_the_father_of(yaser, amir).

is_the_father_of(anas, bader).

is_the_father_of(tarek, khaled).
is_the_father_of(tarek, sara).

```
is_the_father_of(yousef, basem).
is_the_father_of(yousef, rana).

is_the_husband_of(zaid, lailla).
is_the_husband_of(yaser, fatima).
is_the_husband_of(anas, rima).
is_the_husband_of(tarek, safaa).
is_the_husband_of(yousef, lora).
is_the_husband_of(basem, sara).
is_the_mother_of(X, Y):-
        is_the_husband_of(Z, X),
        is_the_father_of(Z, Y).

female(X):-
        (is_the_father_of(_, X),
        not(man(X)));
       (is_the_husband_of(_, X),
        not(is_the_father_of(_, X))).

parent(X,Y):-
        is_the_father_of(X, Y);
        is_the_mother_of(X, Y).
```

```
is_the_brother_of(X,Y):-
        is_the_father_of(Z, X),
        is_the_father_of(Z, Y),
        male(X),
        male(Y),
        X \==Y.

man(X):-
        male(X).

is_the_son_of(X, Y):-
        male(X),
        parent(Y, X).

is_the_daughter_of(X, Y):-
        female(X),
        parent(Y, X).

is_the_mother_in_law(X, Y):-
        (is_the_husband_of(Z, Y),
        is_the_mother_of(X, Z));
```

```
        (is_the_husband_of(Y, Z),
        is_the_mother_of(X, Z)).

is_the_father_in_law(X, Y):-
        (is_the_husband_of(Z, Y),
        is_the_father_of(X, Z));
        (is_the_husband_of(Y, Z),
        is_the_father_of(X, Z)).

is_grandson(X, Y):-
        (is_the_son_of(Z, Y),
        is_the_son_of(X, Z));
        (is_the_daughter_of(Z, Y),
        is_the_son_of(X, Z)).

is_granddaut(X, Y):-
        (is_the_son_of(Z, Y),
        is_the_daughter_of(X, Z));
        (is_the_daughter_of(Z, Y),
        is_the_daughter_of(X, Z)).
```

```
small_child(X):-
        (is_the_son_of(X, Y),
        is_the_son_of(Y, Z),
        male(Z));
        (is_the_daughter_of(X, Y),
        is_the_son_of(Y, Z),
        male(Z)).

is_uncle_of(X, Y):-
        is_the_father_of(Z, Y),
        is_the_brother_of(X, Z).
```

3. أحجية الثمانية وزراء (8-Queens):

تتحرك قطعة الوزير في لعبة الشطرنج بكافة الاتجاهات، والمطلوب كتابة برنامج بلغة البرولوغ لتحديد أماكن ثمانية وزراء دون وجود تقاطعات بينهم.

الحل:

من الأحاجي المسلية والصعبة، أحجية الثمانية وزراء في لعبة الشطرنج، أو كما تسمى في اللغة الإنكليزية (8-Queens)، حيث تسمى قطعة الملكة لدينا بالوزير. يطلب وضع ثماني قطع متشابهة على رقعة الشطرنج القياسية، وكل قطعة يمكنها التحرك بكافة الاتجاهات بشكل مشابه لحركة قطعة الوزير، وبحيث لا تتقاطع أي قطعتين. تطورت خوارزميات إيجاد حل هذه المسائل على اعتبار

وجود (n) وزير على رقعة شطرنج بقياس (n*n)، وقد كانت أفضل وأسرع تلك الخوارزميات هي المعتمدة على التعقب الخلفي. علماً أن الحلول التي نفذت حتى الآن هي من أجل عدد وزراء لا يتجاوز (26) وزيراً فقط، وهذا العدد في تغير مستمر حسب قدرات الحواسيب. كان أول طرح لهذه المسألة عام (1848) من قبل (Max Bezzel). قام العديد من الرياضيين مثل (Gauss) بحل المسألة، كما قام (Franz Nauck) عام (1850) بتوسيع المسألة إلى (n) وزيراً. وفي عام (1874) اقترح (Gueenther) حلاً باستخدام محددات المصفوفات. يوجد من أجل مسألة الثمانية وزراء (92) حلاً، لكن عدد الحلول الفريدة هي (12) حلاً.

يعتمد الحل على تقسيم رقعة الشطرنج إلى أسطر وأعمدة مرقمة من الواحد وحتى الثمانية، كما هو موضح في الشكل(7-6)، حيث يظهر أحد الحلول الممكنة. تكمن فكرة الحل في البدء بمسح (Scan) جميع الخيارات الممكنة لتوضع الوزراء وإظهار الحلول واحداً تلو الآخر حسب رغبة المستخدم. تبدأ عملية المسح بوضع الوزير الأول في السطر الثامن والعمود الأول، ثم اختبار وضع الوزير الثاني في السطر السابع والعمود الأول، فإذا وجد تقاطع، اختبر العمود الثاني فالثالث وعند عدم وجود تقاطع مع الوزراء السابقين، يتم تثبيت الوزير والانتقال للوزير الثالث في السطر السادس. أيضاً تتم عملية الاختبار من العمود الأول حتى العمود الذي يحقق عدم التقاطع. وهكذا تتم العملية بشكل تكراري، فإذا لم يحقق أي من الأعمدة عدم وجود تقاطع فإنه يتم الانتقال للسطر السابق وتحريك الوزير إلى عمود آخر يحقق عدم التقاطع.

	1	2	3	4	5	6	7	8
1				Q				
2		Q						
3							Q	
4			Q					
5						Q		
6								Q
7					Q			
8	Q							

الشكل(7–6) تقسيم رقعة الشطرنج

يفضل في مسائل البحث صياغة عدة طلبات هي فضاء البحث الذي يمثل فضاء الحالات (State Space)، والحالة الابتدائية (Initial State)، والتابع اللاحق (Successor Function)، وشرط الاختبار (Goal Test)، وتكلفة الحل (Cost) أحياناً. في مسألة الثمانية وزراء فإنه يمكن القول إن الحالات تمثل توزيعاً ما للوزراء على رقعة الشطرنج، وبالتالي سيكون عدد الحالات الكلي (64^8) حالة. أما الحالة الابتدائية فهي عدم وجود أي وزير على الرقعة. ويكون التابع اللاحق هو إضافة وزير إلى الرقعة. إن شرط الاختبار هو عدم وجود تقاطعات بين الوزراء. يمكن تخفيض فضاء البحث بشكل كبير باعتبار حجز سطر لكل وزير، وهذا ما سنعتمده برمجياً.

يتم برمجياً تعريف قائمة من ثمانية عناصر، وكل عنصر عبارة عن ثنائية من رقمين بينهما إشارة (//)، تعبر عن رقم السطر والعمود لكل وزير. وبما أنه حتماً كل سطر فيه وزير واحد، وكل عمود فيه وزير واحد، لذلك يتم في البداية حجز سطر لكل وزير مع عدم تحديد العمود. فعلى سبيل المثال يتوزع الوزراء في أحد الحلول الممكنة بالشكل:

Q = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1] ;

هذا الحل مبين في الشكل(7-6)، حيث أن إحداثيات الوزير الأول وفق الثنائية (1/4)، هي السطر الأول والعمود الرابع. وإحداثيات الوزير الثاني وفق الثنائية (2/2)، هي السطر الثاني والعمود الثاني. وإحداثيات الوزير الثالث وفق الثنائية (3/7)، هي السطر الثالث والعمود السابع. وهكذا حتى الوزير الثامن.

يبدأ برنامج البرولوغ بمرحلة التأهيل الابتدائي بتعريف معلن الحقيقة (chess_board/1) بهدف تعريف قائمة من ثمانية عناصر، وكل عنصر عبارة عن ثنائية تعبر عن رقم السطر والعمود لكل وزير، وبالتالي يتم في البداية حجز سطر لكل وزير، لتبقى المشكلة هي في تحديد الطرف الثاني في الثنائية أي تحديد رقم العمود، والذي وضع كمتحول مجهول باستخدام (_).

المعلن الأساسي هو القاعدة (eight_queens/1)، التي تأخذ وسيطاً واحداً هو قائمة الإحداثيات، لتقوم بتخصيص كل وزير بسطر عبر استدعاء المعلن (chess_board/1)، واستدعاء المعلن (arrangement_queens/1)، الذي يقوم بتخصيص كل وزير بعمود، أي حل المسألة.

يعرف المعلن (arrangement_queens/1) كحقيقة لإنهاء شرط العودية، وكعلاقة تقوم باستدعاء المعلن نفسه بشكل عودي، بالإضافة للمعلن مسبق التعريف (member/2)، والمعلن (right/2).

يهدف المعلن (member/2) إلى تشكيل حلقة فحص للمرور على جميع الأعمدة، حيث تتغير قيمة المتحول (Col) من الواحد وحتى الثمانية. لذلك فإن أول اختبار

يهدف المعلن (right/2) إلى فحص عدم وجود تقاطعات بين الوزراء. يفحص المعلن (right/2) فقط التقاطعات العمودية والقطرية لأنه لا يمكن أن

توجد تقاطعات أفقية بسبب تخصيص كل وزير بسطر. أيضاً المعلن (right/2) يعمل بشكل عودي لأنه يفحص احتمال وجود تقاطع بين الوزير الحالي وبقية والوزير السابق، ثم استدعاء العودية لفحص احتمال وجود تقاطع بين الوزير الحالي وبقية الوزراء السابقين. في بداية المعلن (right/2) تتم عملية فحص التقاطع العمودي بفحص تساوي رقم عمود الوزير الحالي مع رقم عمود الوزير الآخر، ثم تتم عملية الفحص القطري بالاتجاهين.

فيما يأتي برنامج البرولوغ الذي يحل هذه المسألة:

```
chess_board([1/_, 2/_, 3/_, 4/_, 5/_, 6/_, 7/_, 8/_]).
eight_queens(Q):-
        chess_board(Q),
        arrangement_queens(Q).
arrangement_queens([]).
arrangement_queens([Row/Col|Rest]):-
        arrangement_queens(Rest),
        member(Col, [1, 2, 3, 4, 5, 6, 7, 8]),
        right(Row/Col, Rest).
right(_, []).
right(Row/Col, [Row1/Col1|R_C]):-
        Col =\= Col1,
        Col1-Col =\= Row1-Row,
        Col1-Col =\= Row-Row1,
        right(Row/Col, R_C).
```

يمكن طلب بعض حلول المسألة بالاستعلام الآتي:

```
?- eight_queens(Q).
Q = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1] ;
Q = [1/5, 2/2, 3/4, 4/7, 5/3, 6/8, 7/6, 8/1] ;
Q = [1/3, 2/5, 3/2, 4/8, 5/6, 6/4, 7/7, 8/1] ;
Q = [1/3, 2/6, 3/4, 4/2, 5/8, 6/5, 7/7, 8/1] ;
Q = [1/5, 2/7, 3/1, 4/3, 5/8, 6/6, 7/4, 8/2] ;
Q = [1/4, 2/6, 3/8, 4/3, 5/1, 6/7, 7/5, 8/2] ;
Q = [1/3, 2/6, 3/8, 4/1, 5/4, 6/7, 7/5, 8/2] ;
Q = [1/5, 2/3, 3/8, 4/4, 5/7, 6/1, 7/6, 8/2] ;
Q = [1/5, 2/7, 3/4, 4/1, 5/3, 6/8, 7/6, 8/2] ;
Q = [1/4, 2/1, 3/5, 4/8, 5/6, 6/3, 7/7, 8/2] ;
Q = [1/3, 2/6, 3/4, 4/1, 5/8, 6/5, 7/7, 8/2] .
```

من الصعوبة تتبع تنفيذ البرنامج حتى النهاية، ولكن كخطوات أولية فإنها تكون بالشكل الآتي:

عند استدعاء المعلن (chess_board/1)، تصبح قيمة المتحول (Q):

```
Q = [1/_, 2/_, 3/_, 4/_, 5/_, 6/_, 7/_, 8/_]
```

أما عند استدعاء المعلن (arrangement_queens/1)، فيتم تعديل المتحولات لتصبح:

```
Row = 1
Rest = [2/_, 3/_, 4/_, 5/_, 6/_, 7/_, 8/_]
```

وعند استدعاء المعلن (arrangement_queens/1)، عودياً كأول معلن ضمن العلاقة فإن المتحولات تصبح:

Row = 2

Rest = [3/_, 4/_, 5/_, 6/_, 7/_, 8/_]

وهكذا حتى الوصول إلى القيم الآتية:

Row = 8

Rest = []

الآن باستدعاء ((arrangement_queens(Rest)، وبما أن قيمة (Rest = [])، فإنه يتم استدعاء الحقيقة ((arrangement_queens([]) التي تمثل شرط إنهاء العودية.

والآن يتم الانتقال إلى المعلن (member) لإجراء حلقة مسح لكافة الأعمدة واختبار وجود تقاطعات مع الوزراء في الأسطر السابقة.

تكون قيم المتحولات حالياً:

Row = 8

Rest = [ ]

Col = 1

يستدعى المعلن (right) لإجراء عملية فحص التقاطعات، وبما أن قيمة المتحول (Rest = [])، فإنه يتم استدعاء الحقيقة ((right(_, []) التي تمثل شرط إنهاء العودية وعدم القيام بأي عمل.

الآن تتم عملية العودية للقيم المخزنة في مرحلة سابقة؛ وهي:

Row = 7

Col = Col1 = 1

Row1 = 8

R_C = [ ]

وبما أن (Col = Col1)، فإن المعلن (right) لن يتحقق، وبالتالي يتوجب القيام بالانتقال لفحص العمود الثاني بزيادة قيمة المتحول (Col = 2)، ولكن هذا أيضاً لن يتحقق من قبل المعلن (right) لوجود التقاطع. أيضاً تزداد قيمة المتحول (Col = 3)، حيث لا يوجد تقاطع ما يؤدي لتحقق المعلن (right)، وإعادة التكرار من السطر السادس. تستمر العملية حتى الحصول على أول حل وإظهاره للمستخدم.

4. أحجية الأوعية (The Water Jugs):

مسألة الأوعية المتعددة (The Water Jugs Problem)، من الأحاجي المسلية، والمطروقة بأساليب مختلفة كتسلية منزلية، حيث يطلب الحصول على كمية محددة من الماء دون استخدام أي أداة قياس. ويسمح باستخدام عدة أوعية ذات أحجام متنوعة للقيام بعمليات مثل الملء والتفريغ والسكب.

يوجد العديد من المسائل المتشابهة، مثل:

1) لديك وعاءان فارغان سعتهما (4, 3 liter)، والمطلوب الحصول على كمية لترين فقط في أحد الوعاءين، علماً أن الوعاء الثالث فيه كمية كبيرة من الزيت دون معرفة الحجم. تتضمن خطوات الحل عمليات ملء وتفريغ وسكب.

2) لدى أحد بائعي الزيت وعاءين فارغين بحجم (7, 3 liter)، ووعاء مملوء بالزيت بحجم (10 liter). أراد أحدهم شراء كمية (5 liter)، على أن تتواجد ضمن وعاء (7 liter). لا يوجد لدى

البائع أي وسيلة قياس ويسمح له القيام بعمليات مبادلة للزيت بين الأوعية للحصول على (5 liter) في الوعاء متوسط الحجم. تتضمن عمليات مبادلة الزيت عملية إفراغ وعاء بشكل كامل ، وعملية ملء وعاء بشكل كامل، وعملية سكب من وعاء لآخر.

الحل:

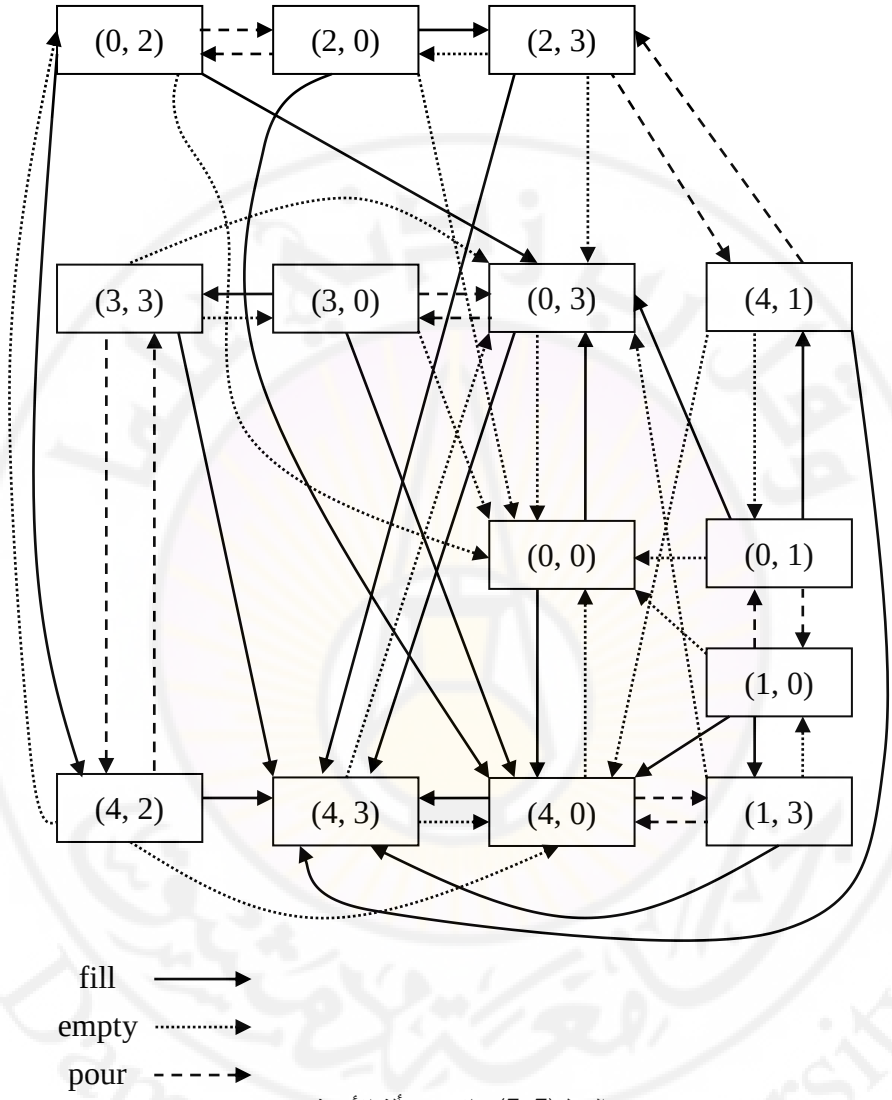
يبدأ الحل اليدوي للمسألة الأولى بملء الوعاء الصغير الأول، ثم تفريغه للوعاء الثاني لتصبح كمية الزيت (0 liter) في الوعاء الأول و (3 liter) في الوعاء الثاني. بعد ذلك يتم إعادة ملء الوعاء الأول وسكبه للثاني. بما أن الوعاء الثاني يحتوي من الخطوة السابقة على (3 liter) وينقصه لتراً واحداً، فإن عملية السكب سوف تتوقف بعد امتلاء الوعاء الثاني، ليتبقى في الوعاء الأول لترين من الزيت، وهو المطلوب.

أما الحل اليدوي للمسألة الثانية فيبدأ بملء الوعاء الصغير من الكبير، ثم تفريغه للوعاء المتوسط. تكرر العملية ثانية ومن ثم يتم ملء الصغير لتصبح الحالة الحالية هي (3 liter) في الصغير و(6 liter) في المتوسط و(1 liter) في الكبير. الآن يتم السكب من الصغير إلى المتوسط فيبقى في الصغير (2 liter). يتم تفريغ المتوسط إلى الكبير ثم تفريغ الصغير إلى المتوسط ثم ملء الصغير من الكبير ثم تفريغ الصغير في المتوسط وعندها يصبح في المتوسط (5 liter)، وهو المطلوب.

بالنسبة للحل البرمجي فإننا سنورد حل المسألة الأولى فقط كون عدد خطوات حلها أقل. علماً أن برنامج البرولوغ للمسألتين متشابه بشكل كبير، ولا يتطلب سوى تعديلات بسيطة ليناسب المسألة الثانية.

لصياغة المسألة برمجياً فإن فضاء الحالات يمثل جميع الحالات الممكنة للوعاءين، حيث يكون للوعاء الأول أربعة احتمالات، وللثاني خمسة احتمالات، والحالة هي ثنائية من الاحتمالين، وبالتالي سيكون عدد الحالات الكلي (20) حالة. أما الحالة الابتدائية فهي عدم وجود زيت في الوعاءين. ويكون التابع اللاحق هو إما التفريغ (Empty) أو الملء (Fill) أو السكب (Pour). إن شرط الاختبار هو امتلاء أحد الوعاءين بلترين من الزيت. أما الكلفة فهي وحدة واحدة لكل انتقال من حالة لأخرى، وكلما كانت هنا الكلفة أقل، كلما كانت خطوات الحل أقل.

يبين الشكل(7-7)، جزءاً من فضاء البحث مع وجود عدة خيارات للحل. تم اعتبار كل حالة عبارة عن ثنائية بالشكل (Jug 4, Jug 3).



الشكل(7-7) حالات مسألة الأوعية

يعتمد البرنامج على اعتبار وجود ثلاثة محددات للعقدة هي حالتها والفعل الذي نفذ لتنتقل لحالتها والكلفة التي أوصلتها لحالتها. تستخدم خوارزمية البحث

بالعرض أولاً، مع إمكانية تأهيل خوارزمية البحث بالعمق أولاً بإزالة إشارة التعليق منها ووضعها على خوارزمية البحث بالعرض أولاً.

المعلن (next) هو المعلن المسؤول عن التابع اللاحق، حيث تم تمييز ثمانية حالات؛ هي:

1) إفراغ الوعاء الكبير: الشرط هو أن يكون الوعاء الكبير يحوي كمية ما من الزيت، وتصبح صفراً.
2) إفراغ الوعاء الصغير: الشرط هو أن يكون الوعاء الصغير يحوي كمية ما من الزيت، وتصبح صفراً.
3) ملء الوعاء الكبير: الشرط هو أن يكون الوعاء الكبير غير ممتلئ ، ويصبح ممتلئاً.
4) ملء الوعاء الصغير: الشرط هو أن يكون الوعاء الصغير غير ممتلئ ، ويصبح ممتلئاً.
5) سكب من الكبير إلى الصغير: الشرط هو أن يكون الوعاء الصغير غير ممتلئ ، ويصبح ممتلئاً.
6) سكب من الكبير إلى الصغير: الشرط هو أن يكون الوعاء الصغير غير ممتلئ ، ويصبح الوعاء الكبير فارغاً.
7) سكب من الصغير إلى الكبير: الشرط هو أن يكون الوعاء الكبير غير ممتلئ ، ويصبح ممتلئاً.
8) سكب من الصغير إلى الكبير: الشرط هو أن يكون الوعاء الكبير غير ممتلئ ، ويصبح الوعاء الصغير فارغاً.

فيما يأتي برنامج البرولوغ الذي يحل هذه المسألة:

```
% node(State, Action, Cost).
state(node(State, _, _), State).
action(node(_, Action, _), Action).
cost(node(_, _, Cost), Cost).
initial_node(node([0, 0], start, 0)).
solution([2, _]).
solution([_, 2]).
%%%%%%%%%%
% solve: %
%%%%%%%%%%
solve([Path|_], _, Path):-
        node(Path, Node,_),
        state(Node, State),
        solution(State).

solve([Path|Open], Closed, Solution):-
        node(Path, Node, _),
        state(Node, State),
        \+  member(State, Closed),
        !,
        expand(Path, NewStates),
```

```
        insert_all(NewStates, Open, NewOpen),
        solve(NewOpen, [State|Closed], Solution).

solve([_|Open], Closed, Solution):-
        solve(Open, Closed, Solution).
%%%%%%%%%
% node: %
%%%%%%%%%
node([Node|_], Node, Node).
%%%%%%%%%%%
% expand: %
%%%%%%%%%%%
expand(Path,NewPaths):-
        node(Path, Node, _),
        findall([NewNode|Path],
        next(Node, NewNode), NewPaths).
%%%%%%%%%%%
%  search: %
%%%%%%%%%%%
search:-
        initial_node(Node),
        solve([[Node]], [], Sol),
```

```
        reverse(Sol, Solution),
        show(Solution).
%%%%%%%%%%%%%%%%%%%%%%
% the show function: %
%%%%%%%%%%%%%%%%%%%%%%
show([]):-
        nl.
show([Node|Reset]):-
        state(Node, S),
        action(Node, A),
        cost(Node, F),
        write(S),
        write('  '),
        write(A),
        write('  '),
        write(F),
        write('  '),
        nl,
        show(Reset).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Breadth-First Search Algorithm : %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
insert_all(NewStates, Fringe, NewFringe):-
        append(Fringe, NewStates, NewFringe).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Depth-First Search Algorithm : %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%insert_all(NewStates, Fringe, NewFringe):-
        append(NewStates, Fringe, NewFringe).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Find The Next Successor: %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%1
next(node([X,  Y], _, Cost), node([0, Y], empty(grand),
Newcost)):-
        X > 0,
        Newcost is Cost+1.
%2
next(node([X, Y], _, Cost), node([X, 0], empty(small),
Newcost)):-
        Y > 0,
        Newcost is Cost+1.
%3
```

```
next(node([X, Y], _, Cost), node([4, Y], fill(grand), Newcost)):-
        X < 4,
        Newcost is Cost+1.
%4
next(node([X, Y], _, Cost), node([X, 3], fill(small), Newcost)):-
        Y < 3,
        Newcost is Cost+1.
%5
next(node([X, Y], _, Cost), node([X1, 3], pour(grand, small), Newcost)):-
        Y < 3,
        X >= 3-Y,
        X1 is X-(3-Y),
        Newcost is Cost+1.
%6
next(node([X, Y], _, Cost), node([0, Y1], pour(grand, small), Newcost)):-
        Y < 3,
        X < 3-Y,
        Y1 is X+Y,
```

```
        Newcost is Cost+1.
    %7
    next(node([X, Y], _, Cost), node([4, Y1], pour(small,
grand), Newcost)):-
        X < 4,
        Y >= 4-X,
        Y1 is Y-(4-X),
        Newcost is Cost+1.
    %8
    next(node([X, Y], _, Cost), node([X1, 0], pour(small,
grand), Newcost)):-
        X < 4,
        Y < 4-X,
        X1 is X+Y,
        Newcost is Cost+1.
```

5. لغز آينشتاين (Einstein Problem):

قام العالم الألماني آينشتاين بطرح هذا اللغز مفترضاً أن (98%) من البشر لا يمكنهم حله. يتحدث اللغز عن مجموعة من الأشخاص بصفات محددة ويطلب الإجابة عن أحد الأسئلة، وفيما يأتي نص اللغز.

1) توجد خمسة منازل، لكل منها لون مختلف.

2) يسكن كل منزل شخص من جنسية مختلفة.

3) يفضل كل شخص أن يشرب مشروباً معيناً.

4) يدخن كل شخص سجائر من نوع معين.

5) يحتفظ كل شخص بحيوان أليف.

6) لا أحد من الأشخاص الخمسة يشترك بصفة مع غيره.

أما الحقائق والعلاقات بين هذه المعطيات، فهي:

1) يسكن البريطاني في المنزل الأحمر.

2) يملك السويدي كلباً.

3) يحب الدنماركي شرب الشاي.

4) البيت الأخضر على الجانب الأيسر من البيت الأبيض.

5) مالك البيت الأخضر يشرب القهوة.

6) الشخص الذي يشرب سجائر نوع بولمال لديه طائر.

7) الرجل الذي يسكن في البيت الأوسط يشرب الحليب.

8) يسكن النرويجي في المنزل الأول.

9) مالك المنزل الأصفر يدخن سجائر نوع دانهيل.

10) يسكن مدخن سجائر نوع مارلبورو مجاوراً لمن لديه قطة.

11) الرجل الذي لديه حصان يسكن مجاوراً لمن يدخن سجائر دانهيل.

12) مدخن سجائر نوع وينفيلد يحب شراب الشعير.

13) يسكن النرويجي مجاوراً للبيت الأزرق.

14) يدخن الألماني سجائر نوع روثمانز.

15) مدخن سجائر نوع مارلبورو لديه جار يحب شرب الماء.

والمطلوب، تحديد الشخص الذي يملك السمكة.

الحل:

يوجد خمسة أشخاص، وخمس جنسيات، وخمسة منازل بخمسة ألوان، وخمسة أنواع من الشراب، وخمسة أنواع من السجائر، وخمسة حيوانات أليفة.

فيما يأتي برنامج البرولوغ الذي يحل هذه المسألة:

```
einstein(Houses, Fish_Owner):-
        =(Houses, [[house, norwegian, _, _, _, _], _,
[house, _, _, _, milk, _], _, _]),
        %Houses = [[house, norwegian, _, _, _, _], _,
[house, _, _, _, milk, _], _, _],

        member([house, brit, _, _, _, red], Houses),
        member([house, swede, dog, _, _, _], Houses),
        member([house, dane, _, _, tea, _], Houses),
        iright([house, _, _, _, _, green], [house, _, _, _, _,
white], Houses),
        member([house, _, _, _, coffee, green], Houses),
        member([house, _, bird, pallmall, _, _], Houses),
```

```
        member([house, _, _, _, milk, _], Houses),
        member([house, _, _, dunhill, _, yellow],
Houses),
        next_to([house, _, _, marlboro, _, _], [house, _,
cat, _, _, _], Houses),
        next_to([house, _, _, dunhill, _, _], [house, _,
horse, _, _, _], Houses),
        member([house, _, _, winfield, beer, _], Houses),
        next_to([house, norwegian, _, _, _, _], [house, _,
_, _, _, blue], Houses),
        member([house, german, _, rothmans, _, _],
Houses),
        next_to([house, _, _, marlboro, _, _], [house, _,
_, _, water, _], Houses),
        member([house, Fish_Owner, fish, _, _, _],
Houses).
next_to(X, Y, List) :-
        iright(X, Y, List).
next_to(X, Y, List) :-
        iright(Y, X, List).
iright(L, R, [L | [R | _]]).
iright(L, R, [_ | Rest]) :-
```

```
iright(L, R, Rest).
```

يمكن طلب حل المسألة بالاستعلام الآتي:

```
?- einstein(Houses, Fish_Owner).
```

لنحصل على أن الألماني هو مالك السمكة.

وقل رب زدني علما
جامعة دمشق
Damascus University

قائمة المراجع:

1. A. Esparcia, C. Sharman, "Genetic programming techniques that evolve recurrent neural networks architectures for signal processing". In IEEE Workshop on Neural Networks for Signal Processing, Seiko, Kyoto, Japan, 1996.
2. A. Samuel, "Some studies in machine learning using the game of checkers ", in Computers and thought, Feigenbaum and Feld man, New York, McGrow-Hill, 1963.
3. A. Tasan, M. Gen, "A genetic Algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries". Computers & Industrial Engineering, 62, 755-761, 2012.
4. D. Goldberg, "Genetic Algorithms in search, optimization and machine learning". Addison-Wesley, 1989.
5. D. Meyer, "An evolutionary Algorithm with Applications to statistics". Journal of computational and graphical ststistics, volume 12, number 2, pages 1-17, DOI:10.1198/106186003169.

6. D. Morton, "Game Theory: A Nontechnical Introduction", 1997.

7. D. Waterman, "Generalization learning techniques for automating the learning of heuristics", Journal of Artificial Intelligence I, pp. 121-170, 1970.

8. E. Goldberg, “Genetic Algorithms in Search, Optimization, and Machine Learning “, Addison-Wesley Publishing Company, Inc, 1989.

9. G. Riley, "Expert Systems: Principles and Programming", 3th Edition, 1998.

10. G. Selfridge, U. Neisser, "Pattern recognition by machine, in Computers and thought, Feigenbaum and Feldman, New York, McGrow-Hill, pp. 237- 256, 1963.

11. H. Smith, G. Lea, "Problem solving and rule induction: a unified view", in L. Gregg, Knowledge and acquisition, Hillsdale, N.J. Lawrence Erlbaum, 1974.

12. I. Bratko, "Prolog Programming for Artificial Intelligence". 3rd edition, 2001.

13. I. Bratko, "Prolog Programming for Artificial Intelligence". Addison-Wesley, Reading, Massachusetts, 2th edition, 1990.

14. J. Bewersdorff, L. Logic, and W. Lies, "The Mathematics of Games", 2005.

15. J. Dolinsky, I. D. Jenkinson, and G. J. Colquhoun, "Application of genetic programming to the calibration of industrial robots". Computers in Industry, 58(3):255/264, 2007.

16. J. Grefenstette, "Optimization of control parameters for genetic algorithms". IEEE-SMC, SMC-16, 122-128, 1986.

17. J. Holland, "Adaptation in Natural and Artificial Systems". MIT Press, 1975.

18. J. Schaffer, A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms". In J.J Grefenstette, Proceedings of the 2nd International Conference on Genetic Algorithms, 36-40. Lawrence Erlbaum Associates, 1987.

19. J. Von Neumann, O. Morgenstern, "Theory of Games and Economic Behavior", .

20. K. Levent, A. Efe, " An Introduction to Game Theory", New York University, 2007.

21. K. Pulji´c and R. Manger, "Comparison of eight evolutionary crossover operators for the vehicle routing problem". Journal of Mathematical Communications, 18, 359–375, 2013

22. L. Davis, "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York, 1991.

23. L. Sterling, E. Shapiro, "The Art of Prolog". 2th edition", 1994.

24. M. Brezocnik, J. Balic, and L. Gusel, "Artifcial intelligence approach to determination of flow curve". Journal for technology of plasticity, 25(1–2):1/7, 2000.

25. M. Dresher, "Games of Strategy, Theory and Applications", 1961.

26. M. Wong, "Evolving recursive programs by using adaptive grammar based genetic programming". Genetic Programming and Evolvable Machines, 6(4):421/455, 2005.

27. N. Nilsson, "The Mathematical Foundations of Learning Machines". Morgan Kaufmann, San Mateo, California, 1990.

28. O. Bahbouh, H. Risheh, "Genetic Algorithms Parameters Effects in Finding Optimal Solution", Damascus University Journal for Engineering Sciences, Volume 23, Issue 2, 2007.

29. O. Bahbouh, "Studying the Parameters of Genetic Algorithms and Their Impact on Problems of Finding the Optimal Solution", Prospects for Applied Mathimatics and data Analysis (PAMDA), Volume 01, Issue 02, PP: 28-36, Publishing (ASPG USA), DOI: 10.54216/PAMDA.010203, 2023.

30. P. Blackburn, J. Bos, and K. Striegnitz, "Learn Prolog Now", 2006.

31. P. Deransart, A. Ed-Dbali, and L. Cervoni, "Prolog: The Standard Reference Manual".

32. R. Davis, "Artificila Intelligence: a guide to intelligent systems", Addison Wisley, 2002.

33. R. Manger, "Comparison of eight evolutionary crossover operators for the vehicle routing problem". Journal of Mathematical Communications, 18, 359-375, 2013.

34. R. Poli, N. Langdon and J. Koza, "Genetic Programming: An Introductory Tutorial and a Survey of Techniques and Applications", ISSN: 1744-8050, 2007.

35. R. Poli, W. Langdon, J. Koza, and N. McPhee, "Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications". ISSN: 1744-8050, 2018

36. S. Cheang, K. Leung, and K. Lee, "Genetic parallel programming: Design and implementation". Evolutionary Computation, 14(2):129{156, 2006.

37. S. Chen, "Genetic Algorithms and Genetic Programming in Computational Finance". Kluwer Academic Publishers, Dordrecht, 2002.

38. S. J. Rusell, P. Norvig, "Artificila Intelligence: A Modern Approach", 4th Edition, Pearson, 2020.

39. S. Ruchira, "Using computer algebra to find Nash equilibira", in Proceedings of the 2003 international

symposium on symbolic and algebraic computation, ACM, 2003,

40. SWI Prolog, reference manual.

41. T. Smith, "Artificial Intelligence". Computer Science 165A, 2005.

42. T. Yu, Rick, L. Riolo, and B. Worzel, "Genetic Programming Theory and Practice III", volume 9 of Genetic Programming, Ann Arbor, 12-14, 2005.

43. V. Kumar, R. Panneerselvam, "A Study of Crossover Operators for Genetic Algorithms to Solve VRP and its Variants and New Sinusoidal Motion Crossover Operator", International Journal of Computational Intelligence Research ISSN 0973-1873 Volume 13, Number 7, pp. 1717-1733, 2017.

44. W. Clocksin, C. Mellish, "Programming in Prolog: Using the ISO Standard". 5th edition, 2003.

45. W. Zhou, T. Song, "Multiobjective Vehicle Routing Problem with Route Balance Based on Genetic Algorithm". Discrete Dynamics in Nature and Society, 2013.

46. Y. Azaria, M. Sipper, "GP-gammon: Genetically programming backgammon players". Genetic Programming and Evolvable Machines, 2005.

47. L. Sterling, E. Shapiro, "The Art of Prolog". 2th edition", 1994.

48. سمير كرمان، "محاضرات مقرر الذكاء الصنعي"، جامعة دمشق، كلية الهندسة الميكانيكية والكهربائية، 2023.

49. أسامة بحبوح، "محاضرات مقرر الذكاء الصنعي"، جامعة دمشق، كلية الهندسة الميكانيكية والكهربائية، 2023.

50. م. الحجار، م. عبيد، "الخوارزميات الجينية". إشراف هـ. العرفي، مشروع دبلوم، جامعة دمشق، كلية الهندسة الميكانيكية والكهربائية، 2003.

اللجنة العلمية:

د. م. عبد الرزاق بدوية

د. م. رشا مسعود

د. م. محمد مازن المحايري

المدقق اللغوي:

د. حسين الزعبي

حقوق الطبع والنشر محفوظة لمديرية الكتب والمطبوعات الجامعية